Java PDF (Limited Copy)

Harvey M. Deitel

Java Summary

Comprehensive Java Programming: Learn, Practice, and Master Essential Concepts.

Written by New York Central Park Page Turners Books Club

About the book

The Deitels' renowned "How to Program" series serves as a comprehensive guide to Java programming, catering to both beginners and intermediate learners. This latest edition is updated to align with the Java 2 Platform Standard Edition (J2SE) 1.5, incorporating essential contemporary features such as autoboxing, which automates the conversion between primitives and their corresponding wrapper classes; enumerations, which provide a type-safe way to define a set of named constants; and enhanced for loops, designed for simplified iteration over collections and arrays.

The book employs a Live-Code Approach, presenting programming concepts through complete, functional examples. This method not only highlights syntax but also includes input/output dialogs that facilitate a more intuitive understanding of programming practices. Each chapter is enriched with practical advice on best practices, common pitfalls, and strategies for performance optimization, helping learners build robust coding habits.

Additionally, the inclusion of a CD-ROM provides essential tools and resources, making the book an indispensable reference for anyone keen to master the Java language. Overall, the Deitels' guide not only teaches Java programming but also mentors readers in the art of coding, ensuring they are well-equipped to tackle real-world programming challenges.

About the author

Harvey M. Deitel is a distinguished educator, author, and software developer, well-respected for his pioneering contributions to computer science education over several decades. He is best known for co-authoring influential textbooks, particularly "Java: How to Program," which has become essential for students and professionals in the field. His innovative teaching strategies focus on making complex programming concepts accessible, earning him widespread acclaim in academic circles.

In addition to his writing, Deitel is a co-founder of Deitel & Associates, a company dedicated to corporate training and software development. This venture underscores his commitment to enhancing technology education and ensuring that both aspiring and experienced developers stay updated with industry trends. Through his work, Deitel has significantly influenced the landscape of programming education, helping to cultivate a generation of adept developers in the constantly evolving realm of software engineering. His efforts reflect a broader mission to make programming more approachable and to prepare learners for the challenges of modern technological demands.

Try Bookey App to read 1000+ summary of world best books

Unlock 1000+ Titles, 80+ Topics

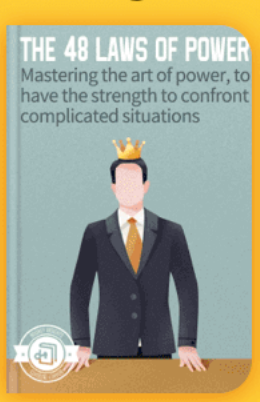
New titles added every week

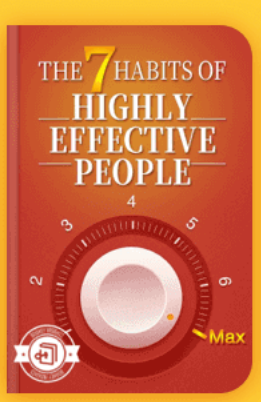
Brand | Leadership & Collaboration | Time Management | Relationship & Communication

ness Strategy | Creativity | Public | Money & Investing | Know Yourself | Positive P

Entrepreneurship | World History | Parent-Child Communication | Self-care | Mind & Spi

Insights of world best books





Free Trial with Bookey

Summary Content List

Multitouch Events and Toasts

Chapter 12: 10 Address Book App: ListActivity, AdapterViews, Adapters, Multiple Activities, SQLite, GUI Styles, Menu Resources and MenuInflater

Chapter 13: 11 Route Tracker App: Google Maps API, GPS, LocationManager, MapActivity, MapView and Overlay

Chapter 14: 12 Slideshow App: Gallery and Media Library Access, Built-In Content Providers, MediaPlayer, Image Transitions, Custom ListActivity Layouts and the View-Holder Pattern

Chapter 15: 13 Enhanced Slideshow App: Serializing Data, Taking Pictures with the Camera and Playing Video in a VideoView

Chapter 16: 14 Weather Viewer App: Web Services, JSON, Fragment, ListFragment, DialogFragment, ActionBar, Tabbed Navigation, App Widgets, Broadcast Intents and BroadcastReceivers

Chapter 1 Summary: Contents

Summary of the Book "Java" by Harvey M. Deitel

Preface and Introduction

The book sets the stage for an in-depth exploration of Java programming, particularly focusing on Android development. It outlines its structure, emphasizing a hands-on approach to learning. The introduction provides context on Android as an operating system, detailing its evolution through various versions, including Froyo, Gingerbread, Honeycomb, and Ice Cream Sandwich, while also introducing the Software Development Kit (SDK) essential for building applications.

Chapter Highlights

1. Introduction to Android

This chapter introduces the Android platform, tracing its historical development and highlighting key features. It discusses the Android Market (now known as Google Play), the SDK, and fundamental concepts in app development, establishing a foundation for future lessons.

2. Android Market and App Business Issues

Expanding on the previous chapter, this section explores best practices in app development, covering important business aspects like registration, effective pricing strategies, and monetization techniques. It also addresses marketing tactics to promote applications in a competitive landscape.

3. Welcome App

A practical introduction to coding begins with setting up the Eclipse Integrated Development Environment (IDE) and creating a simple Android application. This chapter familiarizes readers with the tools and frameworks necessary for development.

4. Tip Calculator App

Building on foundational skills, this chapter walks readers through programming a functional tip calculator. It emphasizes user interface design and coding principles, reinforcing the connection between GUI and backend logic.

5. Favorite Twitter Searches App

Utilizing SharedPreferences for data storage, this chapter guides readers in creating an app that saves and retrieves user preferences, enhancing interactivity through well-implemented UI components.

6. Flag Quiz Game App

Readers design a quiz game that reinforces concepts of asset management and introduces basic animation. This chapter illustrates how to engage users through game mechanics and visual enhancements.

7. Cannon Game App

In this chapter, touch input handling is explored as readers develop a game that manages graphics and sound. This involves understanding user interactions and improving gameplay responsiveness.

8. SpotOn Game App

Focusing on property animation, this chapter teaches how to create more dynamic and visually appealing interactive experiences, a key skill for modern application design.

9. Doodlz App

Here, readers learn to handle multitouch events in a creative drawing application. This chapter emphasizes graphics programming, allowing users to express creativity while understanding advanced input handling techniques.

10. Address Book App

This chapter dives into managing data across multiple activities and SQLite databases, showcasing how to integrate various user interface components to build a functional and efficient contact management tool.

11. Route Tracker App

Utilizing the Google Maps API, this section explains how to use GPS services for real-time location tracking. This practical application showcases the power of integrating location-based services in Android apps.

12. Slideshow App

Readers explore media management as they create an application that accesses galleries, enabling image display within the app—a foundational skill for many types of applications.

13. Enhanced Slideshow App

Building upon the previous chapter, this section enhances the slideshow app by integrating camera functionality and enabling video playback, adding layers to presentation capabilities.

14. Weather Viewer App

This chapter introduces web services and JSON handling, focusing on creating complex interactions with app fragments and action bars for a seamless user experience when displaying weather data.

Additional Chapters

The book further explores practical applications, including voice and audio recording, advanced address book functionalities, and even introduces 3D rendering techniques for more complex app designs.

Wrap-Up

Each chapter concludes with summaries designed to reinforce the concepts covered and solidify understanding. Collectively, the book serves as a comprehensive resource for both beginners and seasoned developers seeking to master Java and Android app development, blending theoretical knowledge with practical skills.

Chapter 2 Summary: Before You Begin

Summary of "Before You Begin"

This introductory section serves as a crucial guide for readers looking to set up their computers to begin developing Android applications using the book "Java" by Harvey M. Deitel.

Font and Naming Conventions help differentiate various components, with on-screen elements presented in a sans-serif bold Helvetica font (for instance, “Project menu”) while Java code appears in a sans-serif Lucida font (like “public” and “class Activity”).

Software and Hardware Requirements outline the necessity for a computer system running on Windows®, Linux, or Mac OS X, with specific development software including the Java SE 6 Software Development Kit, Eclipse 3.6.2 IDE, Android SDK versions 2.2, 2.3.3, and 3.x, as well as the ADT Plugin for Eclipse.

To begin development, **Installing the Java Development Kit (JDK)** is esse ntial, particularly JDK version 6. Detailed installation instructions are provided on the official website.

Next, **Installing the Eclipse IDE**, which is favored for Android development, involves downloading the IDE package for Java Developers and configuring it to utilize JDK 6.

After setting up the IDE, readers will need to focus on **Installing the Android SDK** by downloading it, extracting its content, and remembering that the Android platform must be downloaded separately.

Integration of the Android SDK tools into Eclipse is achieved through the **ADT Plugin for Eclipse**. Installation instructions and troubleshooting tips are also included to assist users.

The next step, **Installing the Android Platform(s)**, requires following specific procedures to configure Eclipse and utilizing the Android SDK Manager for selecting and installing necessary platform packages.

To facilitate app testing across varying devices, **Creating Android Virtual Devices (AVDs)** is crucial. This involves using the Android Emulator to establish AVDs that emulate device specifications.

Optional steps include **Setting Up an Android Device for Development**, which might require a Windows USB driver alongside device-specific drivers to execute apps on real devices.

Lastly, readers are made aware of **(Optional) Other IDEs for Developing Android Apps**, emphasizing that while Eclipse is popular, alternative IDEs and command-line tools are also available for development.

To enhance the learning experience, **Obtaining the Code Examples** is highl ighted. Readers can access code examples by registering for a free account on the designated website, allowing them to follow along with the book's teachings in practical Android app development.

This section effectively equips new learners with the foundational tools and configurations needed to embark on Android app development through the guidance of the book.

Chapter 3 Summary: 1 Introduction to Android

Chapter 1 Summary: Introduction to Android

1.1 Objectives

This chapter provides a comprehensive overview of Android, covering its historical development, including the evolution of the Android SDK, and the Android Market for app distribution. It also introduces fundamental concepts of object-oriented programming and key software tools used in Android app development such as the Android SDK, Java SDK, and the Eclipse IDE. Additionally, it highlights essential documentation and resources for developers, culminating in a practical introduction to a test application that enables users to draw on the screen.

1.2 Android Overview

Launch in 2008, Android rapidly gained traction in the mobile market, particularly in North America. Developed originally by Android, Inc. and later acquired by Google, it operates under the auspices of the Open Handset Alliance. Its open-source model encourages developers to access and modify the source code, contributing to its continual improvement.

1.3 Android Versions

The chapter examines the progression of Android versions, beginning with 2.2 (Froyo) and ending with 3.0 (Honeycomb). Each version brings notable enhancements, particularly in user interface design and overall performance, demonstrating Android's adaptive evolution in response to user and developer needs.

1.4 Features of Android 2.2 (Froyo)

Froyo introduced significant updates including superior performance, better memory management, and features designed for enhanced security, such as password options. New developer tools were also made available, facilitating easier integrations, underscored by the introduction of Cloud to Device Messaging (C2DM).

1.5 Features of Android 2.3 (Gingerbread)

Gingerbread brought a redesigned keyboard and improvements in multitasking and camera accessibility. It also provided developers with more robust communication APIs, enabling better app performance and management.

1.6 Features of Android 3.0 (Honeycomb)

With a clear focus on tablet optimization, Honeycomb enhanced the user interface and functionality tailored for larger screens, marking a pivotal shift in Android's design philosophy to accommodate diverse devices.

1.7 App Distribution via Android Market

This section explains the app acquisition process through the Android Market, outlining the associated developer fees and marketing strategies such as providing free 'lite' versions to encourage user engagement.

1.8 Android Packaging

Android packaging is elaborated upon, detailing how developers can leverage pre-defined class structures to streamline app development, maximizing efficiency.

1.9 Android Software Development Kit (SDK)

The Android SDK is a powerhouse for app creation, equipped with necessary tools optimized for use with Eclipse IDE and emulators for user testing, illustrating the SDK's crucial role in development.

1.10 Object Technology Refresher

This part serves as an introduction to object-oriented programming fundamentals, explaining the relationships between classes, objects, methods, and attributes, while emphasizing core concepts like encapsulation and inheritance essential to software development.

1.11 Test-Driving the Doodlz App

The chapter concludes with a guide to setting up and testing the Doodlz app, illustrating practical experience through the Android emulator. This hands-on approach solidifies understanding of the development tools and processes.

1.12 Deitel Resources

A list of supplementary resources provided by Deitel for those seeking to deepen their knowledge of Android development is presented here.

1.13 Additional Android Development Resources

This section curates a variety of websites, articles, forums, and video channels that serve as valuable tools for Android developers, offering best practices and community support.

1.14 Wrap-Up

The chapter concludes by recapping essential topics, emphasizing the historical context of Android, the key features across versions, foundational Java development skills, the app testing process, and preparation for app submission to the Android Market.

Chapter 4: 2 Android Market and App Business Issues

Summary of "Android Market and App Business Issues"

Objectives Overview:
This chapter aims to guide developers through the essential aspects of creating and distributing Android applications. It covers the key characteristics of successful apps, user interface design, registration procedures for Android Market, various monetization strategies, and insights into broader app distribution channels.

1. Introduction:
The chapter sets the stage for Android app development, emphasizing the importance of user interface (UI) guidelines, registering apps for the marketplace, and effective monetization strategies. These foundational steps are critical for any developer looking to succeed in the competitive landscape of mobile applications.

2. Building Great Android Apps:
Successful Android apps share distinct characteristics: they are innovative, regularly updated, functional, and user-friendly. While gaming apps should entertain and challenge users, utility apps must enhance productivity and provide reliable information. This balance of entertainment and practicality

is vital for user engagement.

3. Android Best Practices:
Developers should adhere to best practices that ensure their apps are compatible with various devices and screens. Following UI guidelines contributes to the app’s performance and responsiveness, enhancing the overall user experience.

4. Registering at Android Market:
To distribute apps on Android Market, developers must pay a one-time registration fee and comply with content policies, allowing them to upload apps without prior approval. This streamlined process supports quick entry into the market.

5. Setting Up a Google Checkout Merchant Account:
For those wishing to sell their apps, establishing a Google Checkout Merchant Account is essential. This involves submitting personal and financial information to facilitate transactions within the marketplace.

6. AndroidManifest.xml File:
The AndroidManifest.xml file is crucial for app functionality and visibility within the market. It details app permissions and features, ensuring proper operation on user devices.

7. Preparing Apps for Publication:

Before launching apps, developers should adhere to a checklist that includes testing, adding icons and labels, managing version control, and digitally signing their apps to enhance security and integrity.

8. Uploading Apps to Android Market:

The upload process consists of submitting the APK file along with relevant screenshots, icons, and descriptions, effectively presenting the app to potential users.

9. Other Android App Marketplaces:

In addition to Android Market, developers have the option to distribute their apps through alternative app stores or directly via their websites, provided they comply with user information policies.

10. Pricing Your App: Free or Fee:

Choosing whether to price an app or offer it for free involves analyzing market competition and exploring monetization strategies, such as in-app purchases or advertisements, to maximize revenue potential.

11. Monetizing Apps with In-App Advertising:

Incorporating advertisements in free apps presents a significant revenue opportunity. This approach allows developers to offer apps at no cost while generating income through ad impressions and clicks.

12. Monetizing Apps: Using In-App Billing:
This strategy enables developers to sell virtual goods and additional content within their apps, with successful examples underscoring the effectiveness of in-app billing as a revenue model.

13. Launching the Market App from Within Your App:
Integrating features that direct users to explore more apps from within their current app can enhance user retention and introduce them to complementary applications.

14. Managing Your Apps in Android Market:
Utilizing the Developer Console, developers can oversee their app’s performance, track reviews, and manage updates effectively to maintain user satisfaction.

15. Marketing Your App:
Effective app marketing strategies can include leveraging social media platforms, email campaigns, and engaging with reviewers to build credibility and attract new users.

16. Other Popular App Platforms:
Expanding one's audience is possible by porting Android apps to other popular platforms, such as iPhone and BlackBerry, allowing developers to

access diverse markets.

17. Android Developer Documentation:

The chapter provides resources and links to essential documentation for further reference, supporting ongoing learning and development in app

Why Bookey is must have App for Book Lovers

30min Content
The deeper and clearer interpretation we provide, the better grasp of each title you have.

Text and Audio format
Absorb knowledge even in fragmented time.

Quiz
Check whether you have mastered what you just learned.

And more
Multiple Voices & fonts, Mind Map, Quotes, IdeaClips...

Free Trial with Bookey

Chapter 5 Summary: 3 Welcome App: Dive-Into® Eclipse and the ADT Plugin

Welcome App Overview: Summary

This chapter serves as a comprehensive introduction to building a simple "Welcome" app using the Eclipse Integrated Development Environment (IDE) with the Android Development Tools (ADT) Plugin. The primary aim is to guide users through the foundational steps of Android app development without the need for coding, focusing instead on visual design and application structure.

Objectives

The key objectives outlined are:

- Gain proficiency with the Eclipse IDE tailored for Android development.
- Create and manage a new Android project.
- Utilize the ADT Plugin to visually design the graphical user interface (GUI).
- Adjust properties of GUI components.
- Develop and run a straightforward Android app on an Android Virtual Device (AVD).

1. Introduction

The chapter begins with an overview of the "Welcome" app, which prominently features a message and two images for visual appeal. This project serves as a hands-on introduction for beginners to familiarize themselves with app development processes.

2. Technologies Overview

Here, the reader learns to navigate the Eclipse platform, covering essential tasks such as creating new projects and the utilization of ImageViews and TextViews, which are fundamental components for displaying images and text in the app's interface. This section also emphasizes the importance of editing the properties of these GUI components to achieve the desired design.

3. Eclipse IDE

Eclipse is established as a pivotal tool for managing Android development, requiring the initial setup of the Java SE Development Kit, Android SDK, and the ADT Plugin. Upon launching Eclipse for the first time, users will encounter the Welcome tab, introducing them to the array of features available in the IDE.

4. Creating a New Project

To kickstart the app development process, users are guided through navigating the menu to create a new project by selecting "File > New > Project" and opting for an Android Project. They will need to input project

details such as the project name, contents, and build target, along with the application name and package name, adhering to Java naming conventions.

5. Building the Welcome App's GUI

The ADT's Visual Layout Editor comes into play, allowing users to create and adjust the main.xml layout file used for the app. The section describes how to choose layout types such as RelativeLayout and LinearLayout and provides a drag-and-drop interface to build the user interface visually.

6. Examining the main.xml File

Readers learn the significance of XML in app development, examining how the GUI is represented structurally in the main.xml file. The discussion includes an exploration of the attributes defining the Visual Layout, including texts and images, contributing to the final design.

7. Running the Welcome App

Once the GUI is constructed, the chapter advises on running the app within an AVD, enabling users to test its functionality and interact with their creation in a simulated Android environment.

8. Wrap-Up

In summary, this chapter highlighted the essential functionalities of the Eclipse IDE, guiding users through the creation of a basic Android app while emphasizing the visual design aspects and customization of component

properties. The subsequent chapter promises to shift focus towards Java programming, essential for enhancing and expanding Android app functionalities.

Chapter 6 Summary: 4 Tip Calculator App: Building an Android App with Java

Tip Calculator App Summary

Objectives

This chapter guides the reader through the development of a Tip Calculator app, focusing on designing its graphical user interface (GUI) with a TableLayout, utilizing Eclipse's ADT Plugin, and implementing Java object-oriented programming (OOP) principles.

1. Introduction

The Tip Calculator app is designed to assist users in calculating tips based on a restaurant bill. It allows users to input the bill amount and calculates the tips and total costs according to common percentages (10%, 15%, 20%) as well as a customizable percentage selected via a SeekBar.

2. Test-Driving the Tip Calculator App

To familiarize you with the functioning of the app, import and run the Tip Calculator project in Eclipse. Key functionalities include entering a bill amount and seeing real-time updates for tips and totals, as well as adjusting the tip percentage using the SeekBar.

3. Technologies Overview

The application leverages key Java OOP concepts—such as classes, interfaces, and inheritance—along with GUI components like EditTexts for user input and SeekBars for adjustable settings.

4. Building the App's GUI

4.1 TableLayout Introduction

A TableLayout is employed to organize the app's GUI components systematically in rows and columns, enhancing the visual structure and ease of navigation.

4.2 Creating the Project

Start by creating the project and configuring the XML layout to integrate the TableLayout, ensuring that it meets the requirements for component organization.

4.3 Adding Components

Components are added to designated TableRows in a logical order, ensuring that each component has the appropriate ID and text settings to function correctly.

4.4 Customizing Components

Components are further tailored to improve user experience through modifications in text alignment, padding, and size adjustments, ultimately ensuring clarity and usability.

4.5 Final XML Markup

The chapter concludes with the full XML code for the app's layout, highlighting various attributes that define its visual presentation.

4.6 strings.xml

An overview of string resources used within the app, enabling easy updates and localization.

5. Adding Functionality to the App

In this section, methods are defined to perform calculations for tips and total amounts based on user input. Event listeners for EditText and SeekBar components are established to enhance interactivity, enabling the app to respond dynamically to user actions and maintaining state across configuration changes, such as screen rotations.

6. Wrap-Up

This chapter detailed the process of creating an interactive Android app using Java, addressing both the design of the GUI and the implementation of underlying functionality. It also introduced concepts of lifecycle management to ensure consistent performance. Future chapters will delve into more advanced functionalities, such as utilizing collections in app development, building upon the foundational work established in this chapter.

Chapter 7 Summary: 5 Favorite Twitter® Searches App: SharedPreferences, Buttons, Nested Layouts, Intents, AlertDialogs, Inflating XML Layouts and the Manifest File

Chapter 7: Summary

Objectives

In this chapter, readers delve into the development of the Favorite Twitter Searches app, a utility designed for users to efficiently store and access their preferred Twitter search strings via customizable tags. This chapter encompasses various facets of Android app development, including user interactions through buttons, the utilization of a ScrollView for content display, the dynamic creation of the graphical user interface (GUI) through XML layout inflation, persistent data management via SharedPreferences, user confirmation through AlertDialogs, and the launching of intents to navigate to web pages.

Introduction

The Favorite Twitter Searches app enhances user experience by enabling quick access to personalized Twitter searches, making it particularly useful

when managing multiple interests or topics without constantly re-entering search terms.

Test-Driving the Favorite Twitter Searches App

The chapter illustrates how to execute and utilize the app in the Eclipse Integrated Development Environment (IDE). It guides readers through the process of adding and managing favorite searches, demonstrating the ease with which users can edit or delete entries, ultimately showcasing the app's user-friendly functionality.

Technologies Overview

Key technological components utilized in the app include:
- **EditText:** for user input.

- **ScrollView:** for displaying a list of stored searches.

- **Button:** for executing actions like adding or removing searches.

- **SharedPreferences:** for managing key-value pairs of user data for persistence.
- **Intents:** for launching external web pages relevant to user searches.

- **LayoutInflater:** for dynamically generating user interface elements.

- **AlertDialog:** for prompting users with confirmation messages on important actions.

Building the App's GUI and Resource Files

The GUI design revolves around a TableLayout defined in XML, allowing structured and responsive user interactions on different screen sizes. The chapter emphasizes the steps required in creating project files and defining the necessary XML attributes to facilitate a cohesive user experience.

Building the App

The app's core functionality resides in a single Activity responsible for managing UI and app components. Critical processes include handling user input through event listeners and managing the storage and retrieval of user searches with SharedPreferences, ensuring that user data is reliably stored across sessions.

AndroidManifest.xml

This section introduces the essential AndroidManifest.xml file, explaining how it specifies the app's package name, versioning details, activity

configurations, and crucial settings that influence the app's behavior, such as suppressing the keyboard on launch.

Wrap-Up

Throughout this chapter, the development of the Favorite Twitter Searches app is meticulously chronicled, emphasizing essential aspects of Android development such as dynamic UI updates, effective persistent data management, and user interaction responsiveness. By laying this groundwork, the chapter prepares readers for future app development ventures, hinting at the exciting prospect of creating a Flag Quiz Game app in Chapter 6.

Chapter 8: 6 Flag Quiz Game App: Assets, AssetManager, Tweened Animations, Handler, Menus and Logging Error Messages

Chapter 6: Flag Quiz Game App

Objectives

In this chapter, you will explore the development of the Flag Quiz Game, focusing on various key components essential for building interactive Android applications. You will learn how to manage resources, manipulate UI elements, and implement features that enhance user experience.

6.1 Introduction

The Flag Quiz Game app challenges users to recognize country flags through a series of multiple-choice questions. Each quiz presents a flag accompanied by three answer options, keeping track of the player's progress and displaying the final results upon completion.

User Interaction Flow

- **Correct Selection**: When a user selects the right answer, the country name appears in green. After a brief pause, the app automatically loads a new flag for the next question.
- **Incorrect Selection**: If the user chooses incorrectly, the app provides

immediate feedback through an animated shake effect on the flag and displays an "Incorrect!" message in red.

- **Quiz Completion**: At the end, players receive a summary of their total guesses and correct answer percentage displayed in an AlertDialog, with an option available to restart the quiz.

Customizations

Users can personalize their gaming experience by choosing the number of answer options—3, 6, or 9—and selecting specific geographical regions from which the flags will be drawn. These options are accessible through the app's menu, allowing for greater engagement.

6.2 Test-Driving the Flag Quiz Game App

To run the app, follow straightforward steps to import the project into Eclipse and launch it. This process verifies that the app functions as intended and allows for immediate testing of features developed.

6.3 Technologies Overview

The foundation of the app is built on several key technologies:

- Flag images are stored in the assets folder and accessed through AssetManager.
- A customizable menu allows users to adjust quiz settings.
- Actions are scheduled using a Handler, which manages delays in displaying the next flag.

- XML animations enhance the user experience by providing visual feedback for incorrect answers.
- Logging is integrated using Android's Log class, which aids in debugging by tracking errors.
- Data structures like ArrayList and HashMap efficiently store and manage the quiz content dynamically.

6.4 Building the App's GUI and Resource Files

The app's graphical user interface (GUI) is crafted using XML to organize components clearly. The main layout, defined in `main.xml`, employs a LinearLayout to arrange visual elements in a user-friendly manner. Resource files, including colors.xml, dimen.xml, and strings.xml, are utilized to standardize UI properties and facilitate the management of string resources.

6.5 Building the App

The core functionality of the app resides within the FlagQuizGame class, which includes:

- **Instance Variables**: Key variables that track game state and interface elements.
- **onCreate Method**: This initializes the activity, sets up the GUI, and starts the quiz.
- **resetQuiz Method**: This prepares the game for a new session, resetting all variables and states.
- **loadNextFlag Method**: Exposes the next flag and dynamically

generates answer choices.

- **submitGuess Method**: Processes user guesses, updates scores accordingly, and navigates the quiz flow.

6.6 AndroidManifest.xml

App Store
Editors' Choice

22k 5 star review

Positive feedback

Sara Scholz

tes after each book summary
erstanding but also make the
and engaging. Bookey has
ding for me.

Fantastic!!! Masood El Toure

I'm amazed by the variety of books and languages Bookey supports. It's not just an app, it's a gateway to global knowledge. Plus, earning points for charity is a big plus!

Fi
Ab
bo
to
m

José Botín

ding habit
's design
ual growth

Love it! Wonnie Tappkx

Bookey offers me time to go through the important parts of a book. It also gives me enough idea whether or not I should purchase the whole book version or not! It is easy to use!

Time saver!

Bookey is my go-to app for
summaries are concise, ins
curated. It's like having acc
right at my fingertips!

Awesome app! Rahul Malviya

I love audiobooks but don't always have time to listen to the entire book! bookey allows me to get a summary of the highlights of the book I'm interested in!!! What a great concept !!!highly recommended!

Beautiful App Alex Walk

This app is a lifesaver for book lovers with busy schedules. The summaries are spot on, and the mind maps help reinforce wh I've learned. Highly recommend!

Free Trial with Bookey

Chapter 9 Summary: 7 Cannon Game App: Listening for Touches and Gestures, Manual Frame-By-Frame Animation, Graphics, Sound, Threading, SurfaceView and SurfaceHolder

Cannon Game App Overview

In this chapter, the reader is guided through the process of creating a dynamic and engaging Cannon Game app. The game's primary objective is to hit a seven-piece target within a strict 10-second time limit. Players control the cannon by tapping the screen to aim and double-tapping to shoot. Successfully hitting segments of the target grants additional time, while inadvertently hitting a blocker reduces the time limit. The game ends with an AlertDialog that displays the player's success or failure.

Objectives

The main goals for this chapter include the development of a user-friendly game app, leveraging a custom SurfaceView subclass for enhanced graphics, and implementing robust touch event handling along with gesture detection. Additionally, sound integration is achieved through the utilization of SoundPool and AudioManager, while improving the lifecycle management

of the Activity ensures a smooth user experience.

Introduction

Players engage by aiming their cannon to strike targets while avoiding blockers that could detract from their score. Hitting the target rewards players by extending their time limit, whereas hitting a blocker brings penalties. The game's conclusion is marked by an AlertDialog, providing clear feedback on the player’s performance.

Test-Driving the Cannon Game App

To ensure users can easily access the Cannon Game, this section provides clear instructions for importing and launching the app within the Eclipse development environment, making it simple for developers to get started.

Technologies Overview

As the development process unfolds, various technologies come into play:

- **String Resources:** Manage localization efficiently with formatting in

strings.xml, enhancing the game's accessibility.

- **Custom View:** Create personalized views, such as CannonView, by extending the SurfaceView class to cater to specific game needs.
- **Sound Management:** Implement sound effects using the SoundPool class, enhancing the immersive experience.
- **Touch Events:** Handle user interactions through the overridden onTouchEvent method of the Activity.
- **Gesture Recognition:** Introduce advanced touch inputs with GestureDetector, allowing for intuitive gameplay.

Building the App's GUI and Resource Files

This section outlines essential steps for setting up the project, including the creation of necessary XML layouts. Sound resources are efficiently organized within the res/raw folder, facilitating easy access during development.

Implementing Game Logic

To create a coherent gaming experience, the chapter delves into the foundational classes that form the game's structure. Key components include:

- **CannonView Class:** This class manages game elements alongside user input, serving as the interface between the player and the game world.
- **Line Class:** Represents the dynamic components of the game by establishing the start and endpoints of various elements.
- **Game Loop Management:** The game updates its state and graphics through a separate thread (CannonThread), ensuring consistent gameplay across devices by synchronizing updates based on timestamps.

Animation and Graphics

To create visually appealing animations, the chapter explains how to manage frame-by-frame animations through a separate thread. This method keeps the user interface responsive, leveraging Canvas and Paint to animate game elements seamlessly while maintaining high performance.

Wrap-Up

In conclusion, this chapter highlights the essential skills acquired while developing the Cannon Game app, laying down a solid foundation for future applications focused on leveraging Android's capabilities for game development. The upcoming chapter will introduce the SpotOn game app,

which will explore property animation for moving images, further expanding developers' knowledge in mobile game design.

Chapter 10 Summary: 8 SpotOn Game App: Property Animation, ViewPropertyAnimator, AnimatorListener, Thread-Safe Collections, Default SharedPreferences for an Activity

Chapter 10: SpotOn Game App Summary

In this chapter, we delve into the development of *SpotOn*, a fast-paced game app designed to test and improve the user's reflexes by requiring them to tap on moving spots before they disappear. As players progress through increasingly challenging levels, they earn points and must manage limited lives, with penalties for missed touches that can lead to game over scenarios.

The chapter begins with a comprehensive guide on importing the *SpotOn* project into the Eclipse development environment, ensuring that players can easily engage with the app's core mechanics. The user interface is built using a series of resource files, including essential layout configurations like `main.xml`, `untouched.xml`, and `life.xml`, along with an updated `AndroidManifest.xml` file. These components collectively create an engaging gameplay experience.

A key feature of *SpotOn* is its animation framework, which utilizes the property animation system introduced in Android 3.0. Unlike previous

methods, this system allows for seamless interaction during animations, enriching the player's experience by ensuring that visual effects do not interfere with gameplay.

The app's primary activity, codified in the `SpotOn` class, oversees the initialization of views and lifecycle management. Meanwhile, the `SpotOnView` subclass encapsulates the game’s internal logic, including the core animations and user interactions.

Several important methods within `SpotOnView` facilitate the game’s operation:
- The constructor establishes necessary game variables and resources.
- The `addNewSpot` method is responsible for creating and configuring new spots on the screen, complete with animations.
- User interaction is captured through methods like `touchedSpot` and `missedSpot`, which adjust scores based on player performance.
- Additional methods manage the overall game state, including resetting scores and tracking lives.

In summary, this chapter illustrates how to harness property animations for an interactive and fluid gaming experience, emphasizing the importance of efficient resource management and responsive user input. These foundational concepts will serve as a springboard for exploring more advanced graphic capabilities in the upcoming chapter on the *Doodlz* app.

Chapter 11 Summary: 9 Doodlz App: Two-Dimensional Graphics, SensorManager, Multitouch Events and Toasts

Chapter 9: Doodlz App - Two-Dimensional Graphics, SensorManager, Multitouch Events, and Toasts

Objectives

In this chapter, you will learn how to:

- Detect user touch events on the screen.
- Process simultaneous touches for enhanced drawing capabilities.
- Utilize SensorManager to implement screen-clearing gestures like shaking the device.
- Ensure safe access to boolean states with AtomicBoolean.
- Customize drawing attributes such as color and width using Paint objects.
- Utilize Path objects to keep track of drawing workflows.
- Display short, user-friendly messages through Toast notifications.

9.1 Introduction

The Doodlz app turns any device screen into an interactive canvas where users can paint using one or multiple fingers. This application allows for color selection and line thickness adjustments through SeekBars, alongside functionalities such as an eraser, screen-clearing options, and the ability to save drawings. A fun feature includes the capability to clear the canvas by simply shaking the device.

9.2 Test-Driving the Doodlz App

While a prior section offered a practical test drive of the app, no further hands-on testing is covered in this chapter.

9.3 Technologies Overview

Doodlz targets Android 3.0, designed to capitalize on its aesthetic improvements. Key technologies and methodologies include:

- The SensorManager, crucial for detecting accelerometer events to clear the screen when the device is shaken.
- Custom Dialogs for adjusting colors and line widths, providing a more refined interface than standard AlertDialogs.
- AtomicBoolean, which facilitates concurrent access management for the dialog states, ensuring safe operations during user interactions.
- The app's drawing features rely on Bitmap and Canvas classes to create engaging graphics.
- Touch events are meticulously captured, enabling users to draw with multiple fingers simultaneously.

9.4 Building the App’s GUI and Resource Files

This section guides you through constructing the essential resources and layouts required for the app:

- **9.4.1 Creating the Project**: Establish a new Android project named Doodlz, entering the prescribed configurations.

- **9.4.2 AndroidManifest.xml**: Ensure the target SDK is set to Android 3.0, leveraging its enhanced features.
- **9.4.3 strings.xml**: Define the string resources that the app will utilize.

- **9.4.4 main.xml**: Integrate a custom DoodleView into the app layout for user interaction.
- **9.4.5 color_dialog.xml**: Design a layout specifically for color selection.

- **9.4.6 width_dialog.xml**: Establish a layout for users to select their desired line width.

9.5 Building the App

In this section, we detail the development of the core application classes:

- **9.5.1 Doodlz Subclass of Activity**: This is the primary activity that manages menu options and responds to accelerometer inputs for functionalities such as clearing the screen.
- **9.5.2 DoodleView Subclass of View**: This class is responsible for capturing user touch inputs and executing the drawing logic on the screen.

9.6 Wrap-Up

In this chapter, you have gained skills in developing an interactive drawing application by employing Android's two-dimensional graphics capabilities, touch event handling, sensor integration, and sound user interface design principles.

Anticipate the next chapter, which will introduce the Address Book app, focusing on effective contact management features.

Chapter 12: 10 Address Book App: ListActivity, AdapterViews, Adapters, Multiple Activities, SQLite, GUI Styles, Menu Resources and MenuInflater

Chapter 10: Address Book App

Objectives

This chapter focuses on teaching key Android development concepts, including extending the ListActivity to create a default ListView, implementing multiple Activity subclasses with explicit Intents, and managing SQLite databases using the SQLiteOpenHelper class. Additionally, it covers inserting, deleting, and querying data with the SQLiteDatabase, displaying results with SimpleCursorAdapter and Cursor, implementing multithreading for database operations, and defining GUI styles via XML and MenuInflater.

10.1 Introduction

The Address Book app provides users with a straightforward interface for viewing and managing contact details. It allows users to scroll through their contacts, edit entries, and delete them when necessary. To ensure that contact information is stored reliably over time, the app makes use of a database for persistent storage.

10.2 Test-Driving the Address Book App

After importing the project into the Eclipse IDE, users can navigate a user-friendly menu that facilitates the addition, editing, and removal of contacts. This menu utilizes standard Android GUI components, including buttons and list views, which enhance the overall usability of the application.

10.3 Technologies Overview

This section outlines the essential technologies powering the app:

- **Manifest**: The `AndroidManifest.xml` file is crucial as it declares each Activity within the app.
- **Styles**: GUI attributes are organized in XML to maintain consistency and ease of design.
- **ListView**: A dynamic list format that displays contacts, extended through Adapter classes for customization.
- **Intents**: These are used for navigating between Activities, facilitating communication within the app.
- **SQLite**: The chosen database engine for storing contact data, supported by utility classes for data management.
- **AsyncTask**: This class is instrumental for carrying out background operations, ensuring that the GUI remains responsive during database transactions.

10.4 Building the GUI and Resource Files

This section highlights the critical aspects of creating the Address Book app's resource files, including:

- **Layouts** for each Activity, defining the visual structure.

- **Menu Resources** that dictate the items available for user interaction.

10.5 Building the App

The app's architecture comprises four primary classes:

1. **AddressBook**: The main Activity housing the ListView of contacts.

2. **ViewContact**: An Activity that presents the details of a selected contact.
3. **AddEditContact**: This Activity enables users to input new contact information or modify existing entries.
4. **DatabaseConnector**: A utility class dedicated to all database operations necessary for managing contact records.

10.5.1 AddressBook Subclass of ListActivity

This subsection explains how to customize the AddressBook class, overriding methods to effectively manage and display contacts within a ListView by utilizing a CursorAdapter.

10.5.2 ViewContact Subclass of Activity

Here, the focus is on displaying individual contact details and providing users with options to either edit or delete the entries.

10.5.3 AddEditContact Subclass of Activity

This part discusses the mechanisms for adding new contacts as well as

Read, Share, Empower

Finish Your Reading Challenge, Donate Books to African Children.

The Concept

BOOKS FOR AFRICA × ×

This book donation activity is rolling out together with Books For Africa. We release this project because we share the same belief as BFA: For many children in Africa, the gift of books truly is a gift of hope.

The Rule

Earn 100 points → **Redeem a book** → **Donate to Africa**

Your learning not only brings knowledge but also allows you to earn points for charitable causes! For every 100 points you earn, a book will be donated to Africa.

Chapter 13 Summary: 11 Route Tracker App: Google Maps API, GPS, LocationManager, MapActivity, MapView and Overlay

Chapter 11: Route Tracker App

Objectives

In this chapter, you will learn how to develop and test the Route Tracker app, which utilizes GPS location data to help users track their movements in real time. You will explore how to integrate the Google Maps API for displaying maps, acquire a unique API key, implement the LocationManager to obtain position and bearing information, and manage device settings to optimize the app's performance.

11.1 Introduction

The Route Tracker app is designed to allow users to monitor their location and direction using an Android device. A simple toggle button enables users to start and stop tracking, with visual feedback provided through changes in text and graphics. The app features a MapView that displays the user's journey, offering various styles, including standard map and satellite views. Upon halting the tracking process, users receive a dialog box that reveals the distance traveled and average speed.

11.2 Test-Driving the Route Tracker App

To get started with the Route Tracker app, follow these initial steps:

- **Importing the App:** Learn how to import the app project into the Eclipse IDE to begin development.
- **Obtaining a Google Maps API Key:** Understand the process for generating a unique API key required for Google Maps functionality, tailored to your development environment.
- **Running the App on an Android Device:** Access detailed instructions for testing the app on a physical device or utilizing the Android Virtual Device (AVD) for simulation.

11.3 Technologies Overview

The Route Tracker app employs several key technologies:

- **ToggleButton:** This interface element allows users to toggle tracking on and off, reflecting the current state visually.
- **MapActivity, MapView, and Overlay:** These classes work together to render Google Maps and draw the user's route effectively.
- **Location Data:** The LocationManager system is utilized to capture real-time GPS coordinates and direction updates.
- **PowerManager and WakeLock:** These components keep the device awake during tracking sessions, preventing it from entering sleep mode.
- **Display Class:** This accesses the device's screen dimensions to ensure the map is appropriately scaled.

11.4 Building the GUI and Resource Files

In the library setup for the Route Tracker project, important elements include:

- **Creating the Project:** Establish the Route Tracker project while specifying build targets and necessary application details.
- **AndroidManifest.xml:** This file guides essential features of the app, addresses library dependencies, customizes the app theme, and requests permissions needed for accessing location services.
- **Route Tracker Layout:** The XML layout file outlines the configuration for the ToggleButton and the FrameLayout that hosts the MapView.

11.5 Building the App

A look into the main components of the app reveals:

- **Class Structure:** The app comprises several key classes, including RouteTracker, BearingFrameLayout, and RouteOverlay.
- **RouteTracker Class:** This principal class manages app activities and interacts with the map interface.
- **BearingFrameLayout Class:** It focuses on the visual orientation and presentation of the MapView.
- **RouteOverlay Class:** Responsible for maintaining and visually representing the tracked route on the map.

11.6 Wrap-Up

The Route Tracker app provides a robust solution for real-time location

tracking using GPS technology and the Google Maps API. Essential features include the ToggleButton for managing user input, efficient integration with location services, and a comprehensive permissions framework. In the forthcoming chapter, we will transition to building a Slideshow app that incorporates multimedia, such as images and audio from the Android library.

Chapter 14 Summary: 12 Slideshow App: Gallery and Media Library Access, Built-In Content Providers, MediaPlayer, Image Transitions, Custom ListActivity Layouts and the View-Holder Pattern

Chapter 14: Slideshow App

In this chapter, we explore the development of a Slideshow app, designed to allow users to craft and control slideshows by incorporating images and music from their devices. The application features a user-friendly main screen that displays a list of slideshows, each showcasing a title and a preview image. Users can select slideshows to play, edit, or delete them, streamlining the management process.

Objectives Covered:

The chapter achieves several key objectives:
- Utilizing Intents and content providers for selecting images and music.
- Implementing launch Intents to return results back to the calling activity.
- Incorporating the MediaPlayer class for playing background music during slideshows.
- Customizing ListActivity layouts and employing the ViewHolder pattern to enhance performance.

- Developing a custom GUI for AlertDialog, enabling the collection of user information.
- Loading images as Bitmaps using BitmapFactory for efficient memory management.
- Creating dynamic image transitions with TransitionDrawable effects.

Introduction:

The Slideshow app serves as a powerful tool for creating personalized visual and auditory experiences. By leveraging existing media on the user's device, the app facilitates a creative outlet that’s both fun and interactive.

Test-Driving the Slideshow App:

Users are guided through the process of importing the Slideshow project into the Eclipse integrated development environment (IDE). The section includes instructions for testing the app's functionalities and transferring necessary media files to an Android Virtual Device (AVD), ensuring smooth operation during development.

Technologies Overview:

The chapter delves into essential technologies that support the app's functionality:

- Built-In Content Providers that facilitate data sharing across different applications.
- Customization of AlertDialog for enhanced user input gathering.
- Creation of custom layouts for ListActivity, allowing for a more tailored user interface.
- The use of startActivityForResult to handle result returns efficiently.
- The role of ArrayAdapter and its extensions in customizing layouts for ListViews.
- Implementation of the ViewHolder pattern to improve ListView performance.
- MediaPlayer for managing audio playback during slideshows.
- BitmapFactory for the efficient loading of images into memory.
- TransitionDrawable for creating smooth transitions between images.

Building the GUI and Resource Files:

Key steps in setting up the project's structure are outlined, including:
- Organizing resource files and implementing standard Android icons.
- Adjustments to the AndroidManifest.xml for essential app configurations.
- Designing layouts tailored to ListView components to enhance user interaction.

Building the App:

The chapter introduces several vital classes that shape the app's architecture:

- **SlideshowInfo**: This class encapsulates the data relevant to each slideshow.
- **Slideshow**: The main interface displaying and managing the various slideshows.
- **SlideshowEditor**: This component allows users to seamlessly add images and audio clips to their slideshows.
- **SlideshowPlayer**: Responsible for controlling the playback of slideshows, integrating both visual and audio elements.

Each class serves a unique purpose, ensuring the app operates smoothly while adhering to best practices such as effective event handling and the ViewHolder optimization for ListView performance.

Wrap-Up:

This chapter underscores the creation of an engaging Slideshow app by tapping into Android’s content providers for data management, allowing for user input customization, enhancing ListView interactions, and utilizing audio playback capabilities. Moving forward, the next chapter will expand on the functionalities of the Slideshow app, promising even greater user experiences.

Chapter 15 Summary: 13 Enhanced Slideshow App: Serializing Data, Taking Pictures with the Camera and Playing Video in a VideoView

Enhanced Slideshow App

Objectives

This chapter outlines how to incorporate advanced features into the Enhanced Slideshow app by enabling video selection, camera capture, and slideshow management. Key elements include:
- Utilizing Intents for selecting videos from the media library.
- Capturing images with the device's rear camera.
- Displaying photos with color effects using SurfaceView and related components.
- Playing videos via VideoView.
- Implementing Serializable objects for saving and loading slideshows.

1. Introduction

The Enhanced Slideshow app expands upon the original Slideshow app by adding file processing and serialization capabilities. This allows users to create memorable slideshows that can be saved and retrieved. Users can

capture new images or select videos, all while enjoying background music during playback.

2. Test-Driving the Enhanced Slideshow App

To explore the functionalities of the Enhanced Slideshow app, users import the project into Eclipse and run it on their Android devices. A guided walkthrough enables the addition of videos and the creation of dynamic slideshows.

3. Technologies Overview

- **File Processing and Object Serialization**: Utilizing Serializable objects allows the app to save slideshow data effectively. Functions such as `writeObject` and `readObject` facilitate this process through Java's ObjectOutputStream and ObjectInputStream.
- **Camera Integration**: The app utilizes the device camera to capture images that can be included in slideshows.
- **Video Playback**: With the VideoView class, users can select and integrate videos, enhancing the visual experience of the slideshow.

4. Building the GUI and Resource Files

This section emphasizes the modification of existing layouts and resource

files to integrate the new functionalities of video selection and camera access, fostering a more interactive user experience.

5. Building the App

Several key classes are introduced to structure the app:
- **MediaItem Class**: Serves as a framework for both images and videos, implementing Serializable to facilitate easy saving and loading.
- **SlideshowInfo Class**: A data structure that retains relevant slideshow details while managing a list of MediaItems.
- **Slideshow Class**: The primary activity responsible for the slideshow's functionalities, including saving and loading instances.
- **PictureTaker Class**: Manages capturing photos and applying visual effects.
- **SlideshowPlayer Class**: Directs the display and playback of both images and videos within the slideshow.

6. Wrap-Up

In summary, this chapter provides a comprehensive look at employing Java I/O for object serialization in storing slideshow data, integrating camera functionalities for photo capture, and enabling video playback in slideshows. The next chapter will shift focus to developing applications for tablets with Android 3.x, as the exploration continues into the creation of a Weather

Viewer app utilizing WeatherBug's web services.

Chapter 16: 14 Weather Viewer App: Web Services, JSON, Fragment, ListFragment, DialogFragment, ActionBar, Tabbed Navigation, App Widgets, Broadcast Intents and BroadcastReceivers

Chapter 16 Summary: Weather Viewer App

Objectives

The primary goal of this chapter is to guide the reader through the creation of a Weather Viewer app tailored for Android 3.x tablets. Key tasks include integrating WeatherBug® web services for real-time weather data, utilizing Android's JsonReader for data processing, employing Fragments for a responsive user interface (UI), creating an interactive app widget, and facilitating user city preference management through broadcasting changes.

Introduction

The Weather Viewer app is designed to provide users with current weather conditions and a five-day forecast for selected cities, making it particularly useful for weather enthusiasts. By leveraging WeatherBug®'s web services, the app ensures that users receive accurate and up-to-date meteorological data.

App Components

1. **User Interface**

- The app features a user-friendly ActionBar that allows easy navigation between two main tabs: current weather conditions and the five-day forecast. Users can personalize their experience by adding new cities or selecting a preferred city through a simple dialog interface.

2. **Fragment Utilization**

- The app employs Fragments to compartmentalize different sections of the UI. For instance, a ListFragment is utilized for displaying a list of cities, while other Fragment types manage individual weather forecasts, enhancing the app's responsiveness and layout flexibility.

3. **App Widget**

- A home screen widget is available, providing users with quick access to current weather updates based on their preferred city without needing to open the app fully. This feature enhances the app’s usability and keeps users engaged.

4. **Data Management**

- The app effectively manages user data by utilizing Android Framework components, such as AsyncTask for executing background tasks and SharedPreferences for storing user preferences conveniently.

Technologies Overview

This chapter emphasizes the critical role of Fragments, the ActionBar, and JSON handling in the app's framework. It provides insights into managing the Fragment lifecycle within Activities, ensuring an uninterrupted user experience even when device orientations change.

Machine Instructions

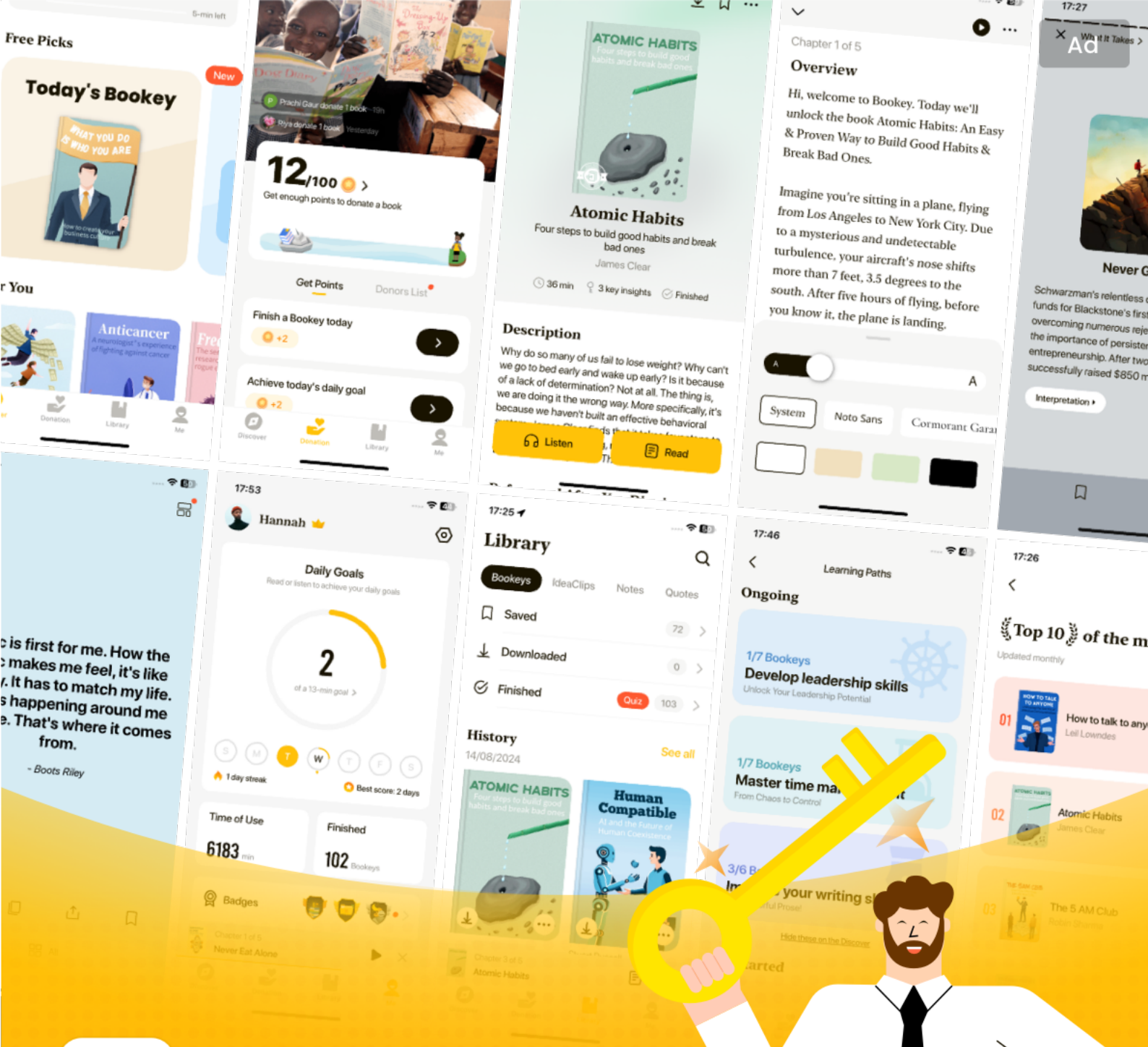
To facilitate development, the app is tested using the Eclipse IDE, with a step-by-step guide on importing project files and running the application. Users are instructed on how to interact with the UI to alter settings and display weather data, particularly with respect to managing city preferences effectively.

Building the App

The app consists of various Java classes, each designed for specific

functionalities:

- **WeatherViewerActivity**: This serves as the main control center, managing UI flow and displaying weather data.
- **Various Fragments**: Each Fragment is responsible for different visual representations of weather data and city lists.



World' best ideas unlock your potencial

Free Trial with Bookey

Download on the App Store

GET IT ON Google Play

Scan to download