RAD

Requirements and Analysis Document for Learn Java

Version: 2.0.1

Date: 26/5-17

Author: Tobias Lindgren, Madeleine Lexén, Hanna Carlsson och Sara Kitzing

This version overrides all previous versions.

1 Introduction

This applications purpose is to get more young people interested in programming and learning how to code. To make the whole process less daunting, and to give an easy start to the journey. The issue at hand is the high threshold to begin coding as a complete beginner. It can be hard to get started and there is a growing demand for programmers. The point is to develop younger peoples interest in programming, and give them a way to get started.

The beneficiary of the application will be youth who might be interested in learning how to code. In the long term it will also benefit companies and educational institutions within computer science. The application will primarily be used as an educational tool during leisure time, but it does not hinder it from being used in school as well.

The application will be similar to a game, as in that it will have several levels in a hierarchy where the user will have to complete a series of steps to reach the next level. These steps will be in form of questions or code assignments.

1.2 Definitions, acronyms and abbreviations

Key: The correct answer
Level: A subcategory within a main category. Contains information about the specific topic and a question on the topic.
Map: The start page, a map over the different main categories
Read more: A page which shows information about basic programming in text form that the user will learn throughout the different worlds. It can be accessed from the menu in all views.
Boss: The last level in a main category which summarizes what the previous levels has taught the user.
Fill in the blanks: A question where the user will get a number of sentences with blanks that they will have to fill in themselves.
Write code: A question where the user will get a task where they need to write code
Toast: A rectangle with text that appears at the bottom of the page

2 Requirements

2.1 User interface

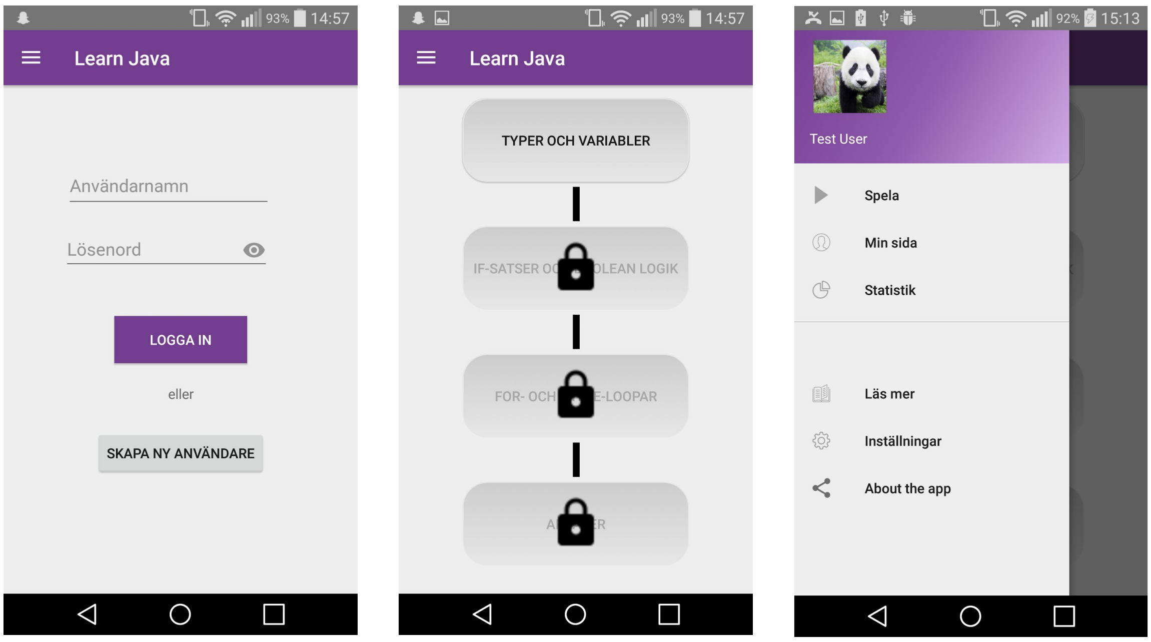
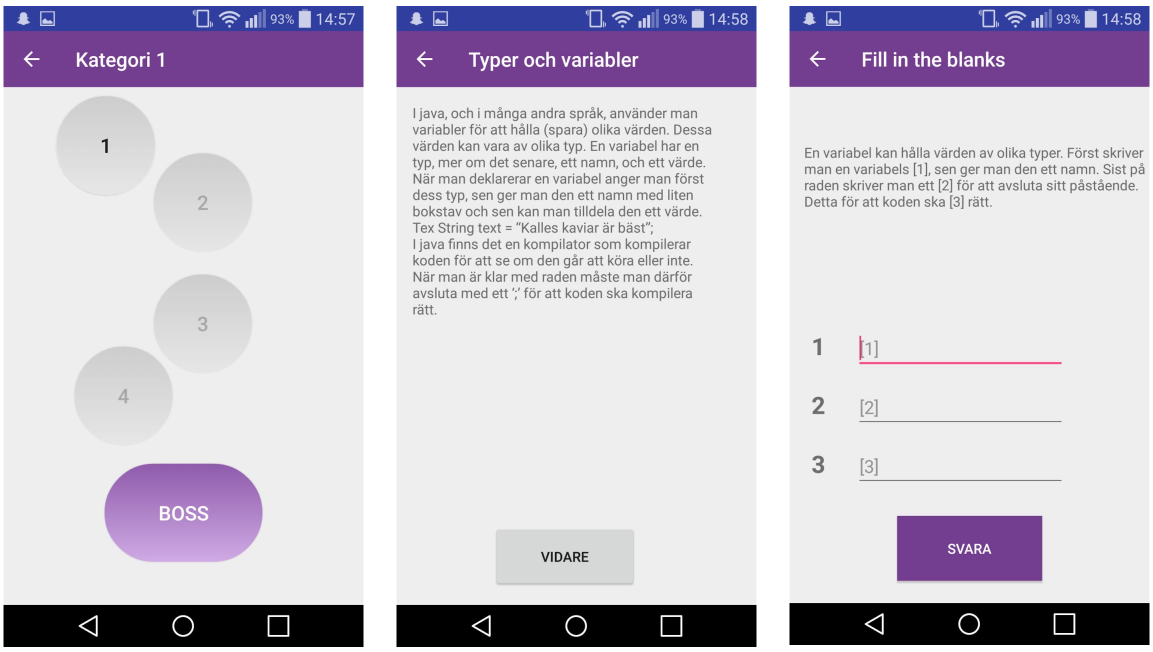


Figure 1. The left view is the start page of the app. Shows the log in page. The middle view is the category view, the categories that are locked is shown by the disabled button and the lock picture. The right view is the navigation drawer.

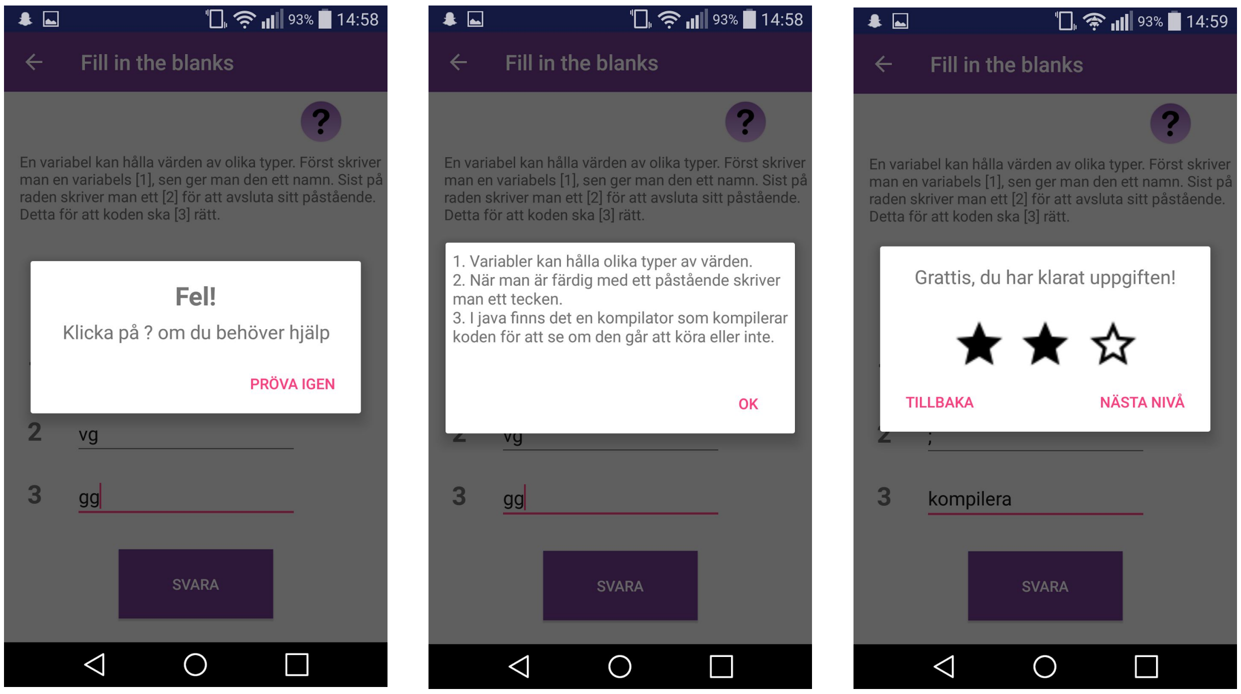


I java, och i många andra språk, använder man variabler för att hålla (spara) olika värden. Dessa värden kan vara av olika typ. En variabel har en typ, mer om det senare, ett namn, och ett värde. När man deklarerar en variabel anger man först dess typ, sen ger man den ett namn med liten bokstav och sen kan man tilldela den ett värde. Tex String text = “Kalles kaviar är bäst”;
I java finns det en kompilator som kompilerar koden för att se om den går att köra eller inte. När man är klar med raden måste man därför avsluta med ett ‘;’ för att koden ska kompilera rätt.

En variabel kan hålla värden av olika typer. Först skriver man en variabels [1], sen ger man den ett namn. Sist på raden skriver man ett [2] för att avsluta sitt påstående. Detta för att koden ska [3] rätt.

[1]
[2]
[3]

Figure 2: The left view is the level view, shown when a category is clicked. The middle view is the question info view, shown before each question. The right view is an example of how a question could look like.


14:58

Fill in the blanks

En variabel kan hålla värden av olika typer. Först skriver man en variabels [1], sen ger man den ett namn. Sist på raden skriver man ett [2] för att avsluta sitt påstående. Detta för att koden ska [3] rätt.

Fel!

Klicka på ? om du behöver hjälp

PRÖVA IGEN

2 vg

3 gg

SVARA

14:58

Fill in the blanks

En variabel kan hålla värden av olika typer. Först skriver man en variabels [1], sen ger man den ett namn. Sist på raden skriver man ett [2] för att avsluta sitt påstående. Detta för att koden ska [3] rätt.

1. Variabler kan hålla olika typer av värden.
2. När man är färdig med ett påstående skriver man ett tecken.
3. I java finns det en kompilator som kompilerar koden för att se om den går att köra eller inte.

OK

3 gg

SVARA

14:59

Fill in the blanks

En variabel kan hålla värden av olika typer. Först skriver man en variabels [1], sen ger man den ett namn. Sist på raden skriver man ett [2] för att avsluta sitt påstående. Detta för att koden ska [3] rätt.

Grattis, du har klarat uppgiften!

★ ★ ☆

TILLBAKA NÄSTA NIVÅ

2 ;

3 kompilera

SVARA

Figure 3: The left view is shown when the answer is wrong. The middle view is shown when question mark button is clicked. The right view is shown when the level is passed, the amount of stars correspond to how well the question was answered.

2.2 Functional requirements

The user needs to log in or create a new user in order to play the game. If the user is new only the first level in the first category will be unlocked, the user can then advance through the categories by unlocking them. This is done by completing every level in previous main categories, which unlocks the next main category. The levels themselves consist of four assignments and at the end of the level there is a boss that the user has to pass. The test is another question but this time you have to write actual Java code.

If the user struggles with the answer to an assignment there is one hint, that guides the user to the answer. If the user still is not able to find the right answer, there will be an option to show the key to the assignment.

The assignments are constructed accordingly; first there is an explanation of the issue at hand, then follows a question testing if the user understood the information. These test can take on one of three forms, which are fill-in-the-blanks, multiple choice and write code.

There will also be a section where the user can read more about what the user is learning about, with a vocabulary and texts giving a brief summary of each category. There will be a possibility for the user to see the statistics as well.

Ordering of the use cases by priority

1. Accessing the game
2. Play game
3. Check answer

2.3 Non-functional requirements

2.2.1 Usability
Since it is a mobile application the user should be able to easily start playing the game, without any explanation. This requires the graphical user interface to be self explanatory.

2.2.2 Implementation
The application is written in Java and should be able to run on all android smartphone devices which run on an OS Lollipop 5.0 or higher. The application will be using a server which compiles code, since it is a requirement for the game.

3 Use cases

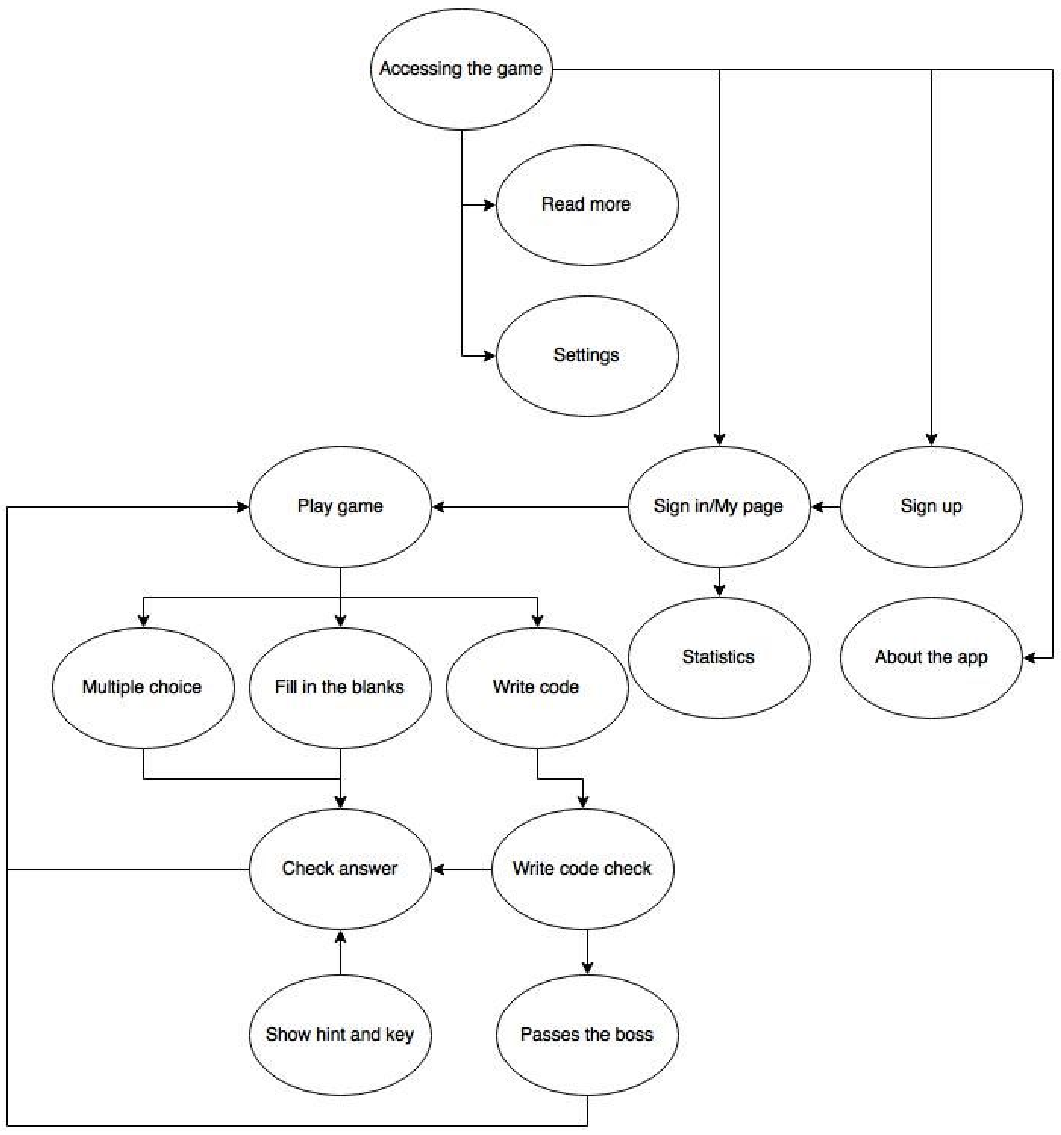


Figure 4: An UML for the applications use cases

3.1 Use case listing

- Accessing the game
    - Sign in
    - Sign up
    - Clicks settings
    - Clicks about the app
    - Click Read more
    - Click statistics
- Play game
    - Check answer
    - Fill in the blanks
    - Multiple question
    - Write code
- Check answer
    - Passes the boss
    - Write code check
    - Show hint and key

The full use cases can be found in the appendix

4 Domain model

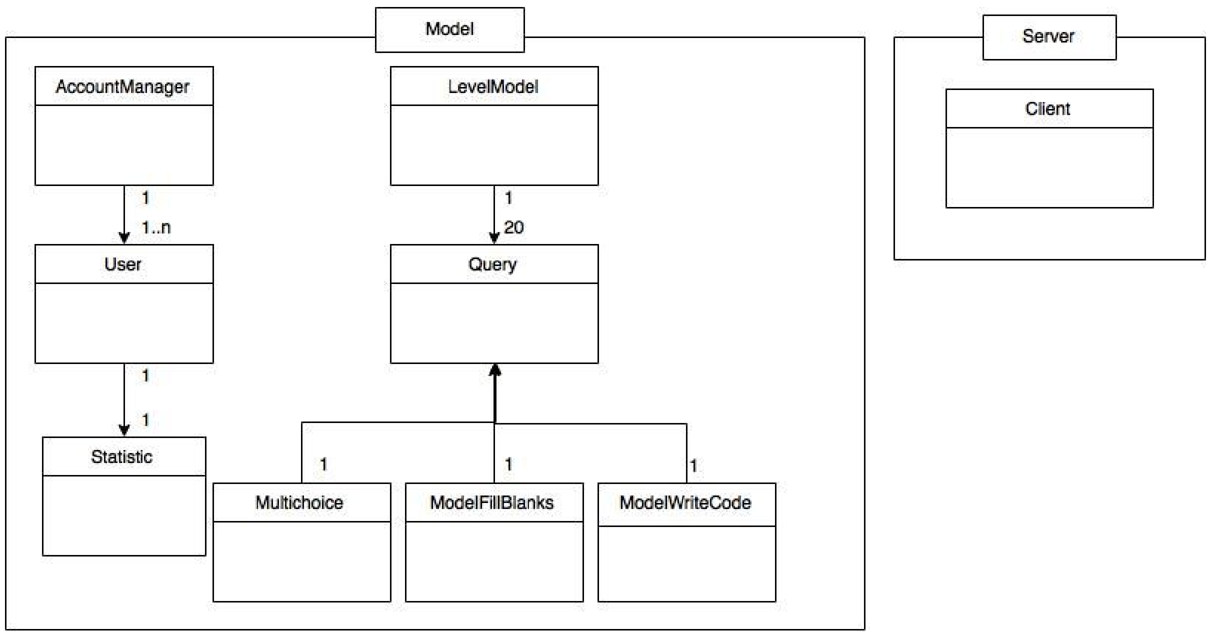


Figure 5: An UML class diagram

4.1 Class responsibilities

LevelModel: This class is responsible for maintaining and storing the hashmap with the level objects.

Query: This class is an abstract superclass to the question classes, and gives them a common interface. It is also responsible for keeping track of the questions, and maintaining them. It also creates the different query objects, each level model object has a query object as an instance variable.
Multichoice: This class handles the logic for the multichoice question, and corrects the users answer.
ModelFillBlanks: Handles the query where the user gets to fill in the words that are missing in a text.
ModelWriteCode: Handles the query where the user gets to write its own code.
AccountManager: Handles the log in and add user functions along with loading and storing the user objects.
User: Handles the updating of the user and holds a statistics object which calls the save statistics methods.
Statistics: Stores the time it takes to finish the question, how many hints needed and if you looked at the key.
Client: Handles receiving the data from the server, compiling the code and sending it back.

6 Appendix

6.1 Use case: Accessing the game

Summary: The user opens the application and logs in
Priority: high
Extends:
Includes: Sign in, Sign up, Clicks settings, Clicks statistics, Clicks about the app, Clicks read more
Participants: The user

Normal flow of events

The user opens the application and signs in/signs up.

	Actor	System
1	Opens the application	
2		Shows the login page
3.1.1	Write username and password	

3.1.2		(Password is correct) Logs in and shows the start page, the main categories
3.1.3		(Password is incorrect) Toast shows the password or username is incorrect
3.2.1	Clicks on sign up	
3.2.2		Shows the sign up page
3.2.3	Writes username, password and chooses a profile picture	
3.2.4		(Username is ok) Logs in and shows the start page, the main categories
3.2.5		(Username if already taken) Toast shows the username is already taken
4		Shows category view

Alternate flow

Flow 2 (Check settings)

The user clicks on settings in the menu

	Actor	System
1	Clicks on settings	
2		Shows the settings page
3.1.1	Drags volume slider	
3.1.2		Volume changes

3.2.1	Clicks on sounds off button	
3.2.2		Sound stops

Flow 3 (Clicks read more)

The user clicks on read more in the menu

	Actor	System
1	Klicks “Read more”-button	
2		Shows read more page

Flow 4 (Clicks statistics)

The user clicks on statistics in the menu

	Actor	System
1	Clicks “Statistics”-button	
2.1		(If logged in) Shows users statistics
2.2		(If not logged in) Toast showing message that you can not access statistics when not logged in

Flow 5 (Clicks about the app)

The user clicks on about appen in the menu

	Actor	System
1	Clicks “About the app”-button	
2		Shows information about the app

Flow 6 (Clicks sign in/my page)

The user clicks on sign in/my page (depending if you are logged in or not)

	Actor	System
1		Shows sign in if not logged in, my page if logged in
2	Clicks on button	
3.1		(If not logged in) See use case *accessing the game (2)*
3.2		(If logged in) Shows my page with user information
3.2.1	Clicks update user-button	
3.2.1.1		Shows update user page
3.2.1.2	Changes username, password and/or user picture.	
3.2.1.3	Clicks update information-button	
3.2.1.4		Updates information
3.2.2	Clicks log out-button	
3.2.2.1		See Normal flow of events (2)

6.2 Use case: Play game

Summary: The user plays the game
Priority: high
Extends: Accesing the game
Includes: Check answer, Fill in the blanks, Multiple choice, Write code
Participants: The user

Normal flow of events

The user clicks on an unlocked level

	Actor	System
1	Clicks on category button	
2.1		(If unlocked) Shows level view
2.2		(If locked) Nothing happens
3	Clicks on level button	
4.1		(If unlocked) Shows level information
4.2		(If locked) Nothing happens
5	Clicks on continue button	
6		Shows question, See *Alternate flow 2, Alternate flow 3, Alternate flow 4*
7	Clicks on submit button	
8		User case *check answer*

Alternate flow

Flow 2 (Fill in the blanks)

User is on a fill in the blanks question

	Actor	System

1		Shows question with numbers as blanks
2	Fills in the answer connected to the number	

Flow 3 (Multiple choice)

User is on a multiple choice question

	Actor	System
1		Shows relevant question with multiple choices
2	Selects a radiobutton	

Flow 3 (Write code)

User is on a write code question

	Actor	System
1		Shows assignment
2	Answer the question by writing code	

6.3

Use case: Check answer

Summary: The system checks if the answer is correct and the user acts upon it.
Priority: high
Extends: Play game
Includes: Passes the boss, write code check, show hint and key
Participants: The user, the server

Normal flow of events

	Actor	System
1		(If no input) Toast shows no input
2.1		(If write code question) See *alternate flow 2*
2.2		Checks if answer is correct
3.1		(If correct) Shows passed level view
3.1.1	Clicks on next level button	
3.1.1.1		(If write code question) See *alternate flow 3*
3.1.1.2		See use case *play game (4.1)*
3.1.2	Clicks on back button	
3.1.3		Shows level map

3.2		(If incorrect) Shows failed level view
3.2.1	Clicks on try again button	
3.2.2		Shows question, see use case *play game (6)*
3.2.3	Clicks on submit button	
3.2.4.1		(If correct) See use case *check answer (2.2)*
3.2.4.2		(If incorrect) See *alternate flow 4*

Alternate flow

Flow 2 (Write code check)

The systems way of handle checking the answer if the query is WriteCode

	Actor	System
1		Creates server-object, sends users code
2		Compiles the code and sends back
3		Checks if answer is correct
3.1		(If correct) See use case *play game (3.1)*
3.1.1		See *Alternate flow 3*
3.2		(If incorrect) Toast shows what the code actual result was

4		See use case *play game (3.1)*

Flow 3 (Passes the boss)

The user passes the last level in a category

	Actor	System
1.1		(If not category 4) Shows level view for next category with toast “You have unlocked the next category”
1.2		(If category 4) Shows category view with toast “Congratulation, you are done with LearnJava”
2		See use case *play game*

Flow 4 (Show hint and key)

The user is unsure of how to solve an objective

	Actor	System
1		Show question view with “?”-button
2	Clicks on hint button	
3.1		(If user is wrong once) Show hint
3.2		(If user is wrong more than once) Show hint and key
5	Clicks on ok	
6		See use case *play game (6)*, but with “?”-button as well