Java Fundamentals

Developing and Testing an Application

ORACLE ACADEMY

Overview

This lesson covers the following topics:

- Demonstrate program testing strategies
- Recognize phases for developing a software application

Program Testing Strategies

- A programmer tests a program many times during the course of its development to ensure it works properly.
- Program testing strategies:
  - Test frequently after each method, or a sequence of methods, are written.
  - Compile the program to ensure it is error-free. If errors appear, correct them.
  - Run the program to observe how the methods make the objects move.
  - Continue to add methods and adjust as necessary.

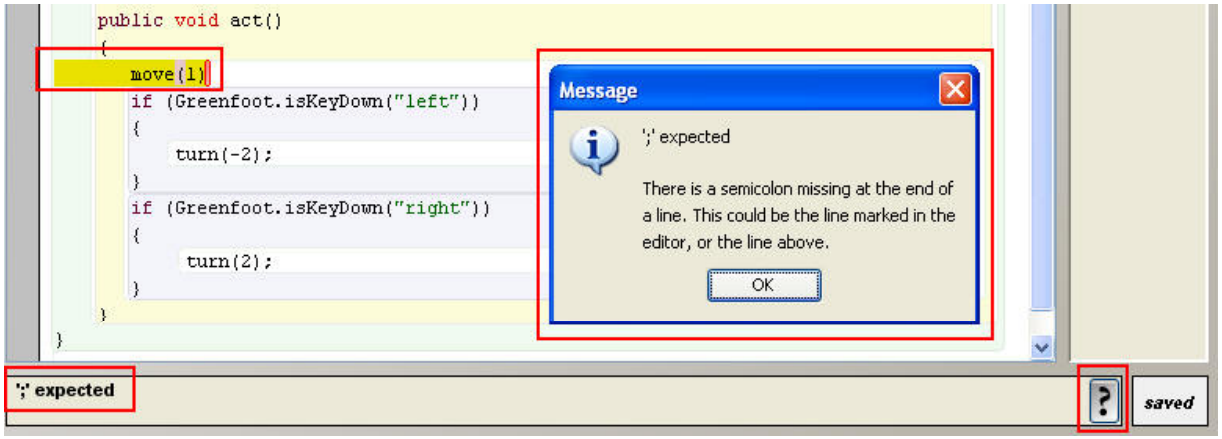
Compilation and Debugging

- Every character in source code counts. One missing or incorrect character could cause your program to fail.
- In Greenfoot, compilation highlights bugs and what is required to correct them. This helps you develop good programming techniques.

Bugs are errors in the syntax of a computer program. To debug a program, the programmer compiles the program and reads any error messages that Greenfoot provides. Then, the programmer corrects those errors in the syntax and re-compiles the program.

Steps to Debug Your Program

1. Click Compile to compile the code.
2. If there are no errors, the message “Class compiled – no syntax errors” displays.
3. If there are errors, the incorrect syntax is highlighted and a message attempts to explain the error.
4. Click the question mark icon to display additional information about the error.



Keys to Recognizing Java Syntax Errors

1	Locate the beginning and end of a method.
2	Ensure all beginning { and ending } braces exist.
3	Ensure all open (and closed) parentheses exist.
4	Ensure all lines of code end with a semicolon.
5	Ensure class names are spelled and capitalized properly.
6	Review all dot notation (i.e., System.out.println).
7	Ensure similar-looking characters are correct (number 1 versus letter i).
8	Ensure all string quotes are double “ not single ‘.

Phases to Develop an Application

1. Analyze the problem to solve or task to perform.
2. Design the solution, which is a game in Greenfoot.
3. Develop the game in Greenfoot.
4. Test the game to ensure it works and meets the requirements of your analysis and design.

Developing a game in Greenfoot follows the same steps as developing a software application.

Analysis Phase

In the analysis phase, determine what problem the game will solve, or the task it will perform, using object oriented analysis.

In object oriented analysis, Java programmers analyze a problem and then create objects to build a system, or more specifically, to solve the problem.

Analysis Phase Tasks

1. Identify a problem to solve.
2. Write a brief statement of scope that states the type of solution (game) that will solve the problem.
3. Gather the target audience's requirements. These are the people who most likely will play your game.
4. Identify and describe objects in the game.
    1. Physical objects (car, person, tree).
    2. Conceptual (“non-physical”) objects (timer that counts down time remaining in the game).
    3. Attributes of all objects, such as color, size, name, and shape.
    4. Operations that the objects perform (move, turn, eat other objects).

Analysis Example

Analysis Item	Description
Problem domain	I want to create a timed game to teach students how to count.
Game player's requirements	It should be easy for 7-8 year olds to play. It requires the player to have a keyboard and mouse.
Objects	1 object named "Duke", the Java mascot; 10 objects for Duke to eat, and a background world that is a light color.
Objects operations	Duke: Move, turn, and eat code. Code: Sit still and be eaten when Duke lands on them. Timer: Count down from 10-1 seconds, then end the game. Background: Do nothing.

Analysis Pre and Post Conditions

Capture information to support testing for:

Item to Test	Example
Pre- and post game conditions.	Variable initialized values versus final values after program execution.
Anticipated run times and comparison run rates given a set of conditions.	Run rates can vary based on computer memory size variance.
Expected results for statement execution counts.	A loop counter of three will produce three new variables.
Numerical representations and limitations.	An integer's maximum value.

Design Phase

- The solution you design will be in the form of a Greenfoot game that your target audience can play.
- Design your game in a textual storyboard that plans the algorithms, or methods, that objects will perform in response to keyboard commands or mouse clicks.

Textual Storyboard Example

This textual storyboard describes a simple game where Duke, the Java mascot, has ten seconds to eat as much Java code as possible.

When the Run button is clicked, Duke can move.

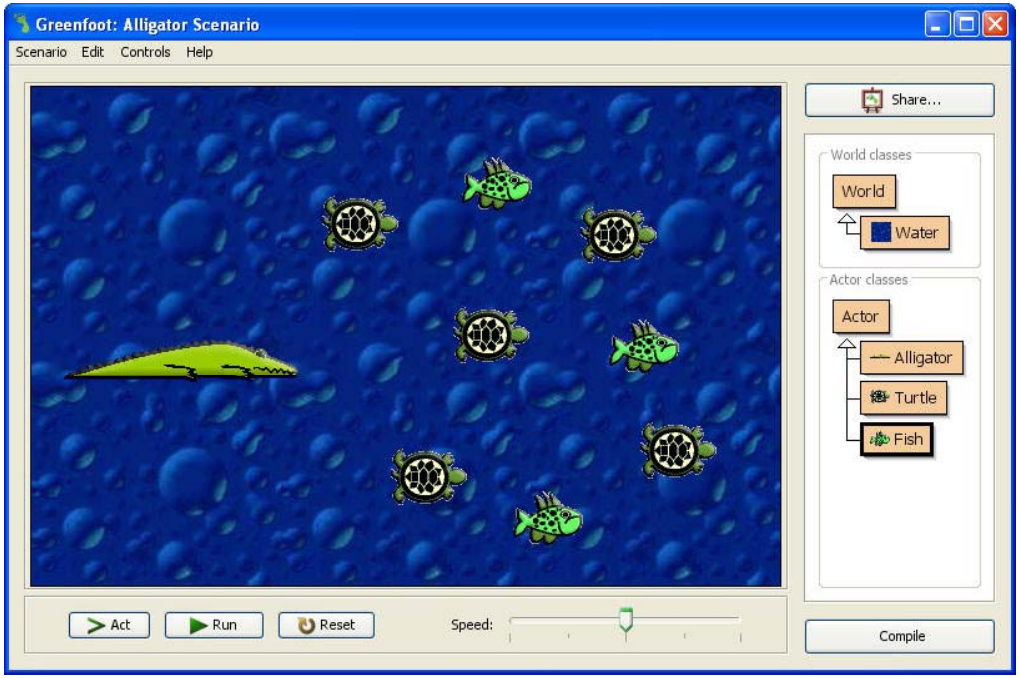
The player uses the arrow keys on the keyboard to control Duke's movements.

When Duke is in the same square as Code, Duke eats Code.

After ten seconds, the game ends and displays how many pieces of Code Duke ate.

Development Phase

After you finalize your storyboard, develop your game in Greenfoot. Refer to your storyboard to determine the methods you need to program.



Testing Phase

- After you write a section of code, compile it, then test it by clicking the Run button in the environment.
- Observe the game then revise the code as necessary.
- For quality assurance purposes:
  - Have other people test your game and give you feedback.
  - Seek people who fit the target audience for your game.
- Write test plans that:
  - Examine pre and post conditions.
  - Compare run time rates and execution counts.
  - Thoroughly test numerical representations and limitations.

Testing Numerical Representations and Limits Example 1

- For example, a banking solution requires precise rounding of numbers.
  - This program could produce incorrect results if the rounding of a number was not set to two digits after a decimal point.
  - The addition of a 1/2 of a cent multiplied by a million customers could produce an expensive programming error.

Testing Numerical Representations and Limits Example 2

- For example, an IF construct in a program expects a positive value of 5 through 9 to then add that value to another variable.
  - The program incorrectly feeds the variable a value of 2.
  - This causes the IF construct to fail and the variable expecting a conditional change will not get the expected amount so the operation of the data structure will be different.
  - This is an example of where the program will execute the result of the conditional construct, even if it is incorrect.

Terminology

Key terms used in this lesson included:

- Bugs
- Documentation

Summary

In this lesson, you should have learned how to:

- Demonstrate program testing strategies
- Recognize phases for developing a software application

Practice

The exercises for this lesson cover the following topics:

- Understanding documentation and testing
- Concept summary and terminology review