Java Fundamentals

3-4

Developing and Testing an Application

ORACLE
Academy
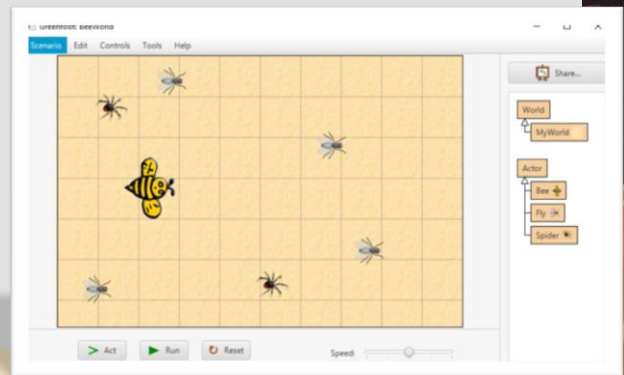
Objectives

- This lesson covers the following objectives:
  - 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
  - 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

Testing is an important aspect of developing software. You are constantly testing your program as you write source code, compile and run. Having a clear testing strategy can greatly add to the quality of your software.

Some aspects of your code will be tested by you, but other aspects will be tested by others. Having other users, especially the ones that the software is aimed at, testing your program, will help you remove errors and increase functionality of your software.

Compilation and Debugging

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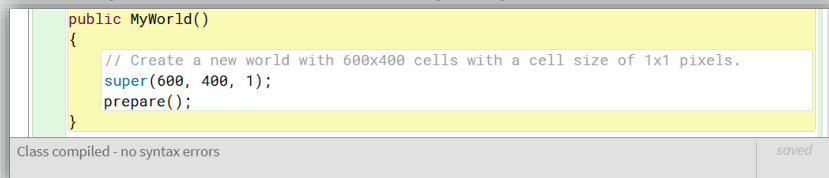- This helps you develop good programming techniques

Bugs are errors in the code of a computer program. To debug a program, the programmer reads any error messages that Greenfoot provides.
Then, the programmer corrects those errors in the syntax. Testing will then move onto the logic in the code.

Remember that successfully compiling software does not mean that it is bug free. It only means that the syntax is correct.

Steps to Debug Your Program

- If there are no errors, the message "Class compiled – no syntax errors" displays
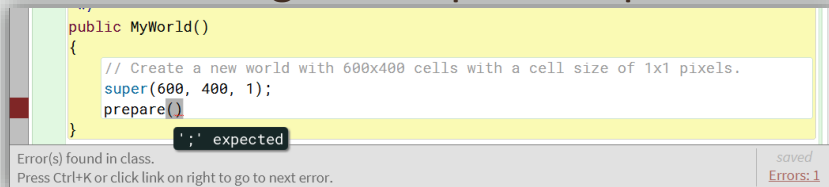
```
public MyWorld()
{
    // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
    super(600, 400, 1);
    prepare();
}
```
Class compiled - no syntax errors   *saved*

- If there are errors, the incorrect syntax is highlighted and a message attempts to explain the error

```
public MyWorld()
{
    // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
    super(600, 400, 1);
    prepare()
}
        ';' expected
```
Error(s) found in class.   *saved*
Press Ctrl+K or click link on right to go to next error.   Errors: 1

ORACLE
Academy

JF 3-4
Developing and Testing an Application

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 '**

ORACLE
Academy

JF 3-4
Developing and Testing an Application

7

Using good code indentation will greatly improve the readability of your code. This makes locating errors like those listed above a lot easier and less time consuming.

Auto-Layout

- A useful function within the Greenfoot code editor is the Auto-Layout feature
- You will find this automatically structures your code and is a great tool to find where your missing brackets are!
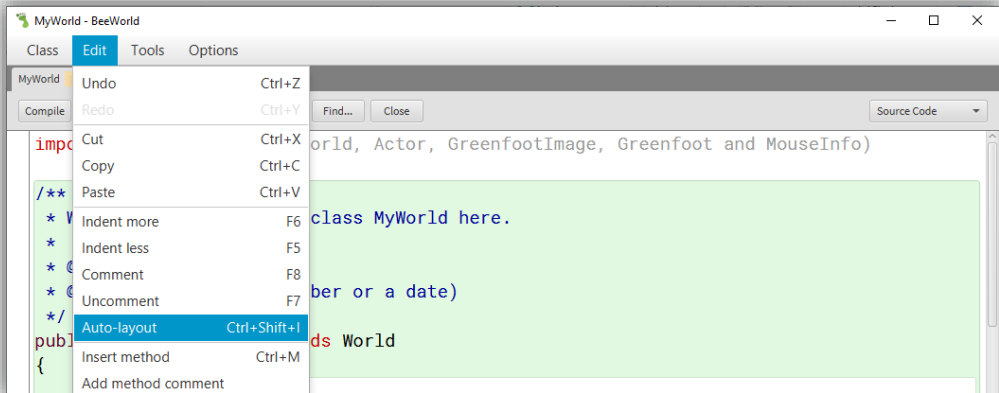
The auto layout will indent code in-between brackets. This demonstrates good program layout techniques to make your code more readable. You could write all your code on one line and Greenfoot wouldn't mind, but trying to find errors in your code becomes very difficult. Also simply trying to read how the code works becomes a huge onerous task.

8

Phases to Develop an Application

1. Analyze the problem to solve or task to perform
2. Design the solution, which is commonly 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
5. Developing a game in Greenfoot follows the same steps as developing a software application

Planning your game before you start coding will save you lots of time. Some simple games will require very little planning, but as the game complexity increases then so does the need for proper planning techniques.

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.
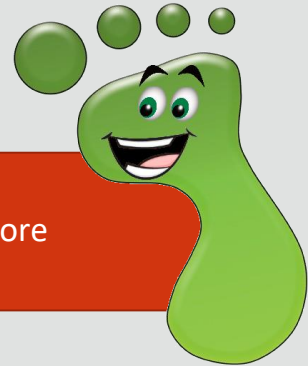
The identification of the objects required in software will help you determine the number of subclasses required under the Actor class. Although we will typically have one level of classes under Actor, in larger programs we may have multiple levels where we have Actor -> subclass->subclass where classes share common fields and methods.

Analysis Phase Tasks

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

ORACLE
Academy

11

Collecting of the information required will better help you plan a solution.

Analysis Example

Analysis Item	Description
Problem domain	I want to create a game to teach students to control a Bee with the cursor keys
Game player's requirements	It should be easy for kids of all ages to play It requires the player to have a keyboard
Objects	1 Bee object that will catch flies Multiple Fly objects 1 Spider object which will catch flies and the Bee 1 World with a light colored background
Objects operations	Bee: Move, turn, and catch flies Fly: Randomly move around the screen Life count: Count down by 1 from 3 every time the Bee is caught by a Spider Background: Do nothing

Defining the actions of an object will give you the basis of the methods and fields required in your classes.

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

Testing can be planned before any coding has started. This has the benefit of having the programmers think about what is going to be tested as they start to code a solution.

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

A good design allows you to think how all of your objects are going to act and interact. It is easy when writing code that does not follow a design to get caught up with only the current problem and not the bigger picture. This can lead to poorly coded solutions.

Textual Storyboard Example

- This textual storyboard on the next slide describes a simple game where you control a Bee to try and catch Flies while avoiding a Spider
- The spider will also catch flies
- You will gain points for every Fly caught and lose a life every time a Spider catches the Bee
- The game ends when you run out of lives

Textual Storyboard Example

- When the Run button is clicked, the bee will continually move forward

- The player uses the arrow keys on the keyboard to control the Bee's left and right movements

- When the Bee is in the same square as a randomly moving Fly, it is then caught and removed from the game and another Fly is added

A textual storyboard is complete when you could give it to any programmer and they would produce very similar results to others. If they all created completely different solutions, then it was the storyboard that was incomplete.

You can test your storyboard by giving it to three people and then have them explain back to you how the game works. If there are big differences in their explanations, then your storyboard requires additional information.

Textual Storyboard Example

- A Spider will move randomly around the screen
- If the Spider catches a Bee then the user loses a life
- If it catches a Fly then it's removed from the game
- Game ends when the user has no lives remaining

A textual storyboard is complete when you could give it to any programmer and they would produce very similar results to others. If they all created completely different solutions, then it was the storyboard that was incomplete.

You can test your storyboard by giving it to three people and then have them explain back to you how the game works. If there are big differences in their explanations, then your storyboard requires additional information.

Development Phase

- After you finalize your storyboard, develop your game in Greenfoot
- Refer to your storyboard to determine the methods you need to program
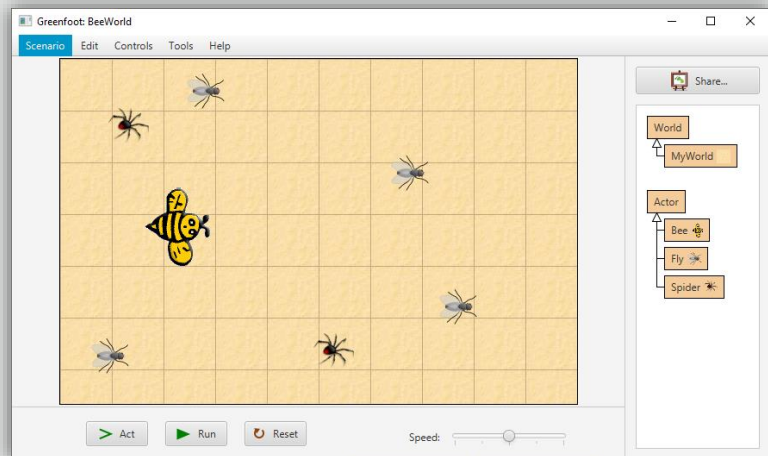
JF 3-4
Developing and Testing an Application

18

On this slide we have not written any code. We have only created the classes we require and added instances of these classes to our scenario to get a feel on how the program is going to look.

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 the program in small stages allows you to pinpoint errors easier as you have a better idea on where they probably reside. If you wrote your whole program before testing it would take a lot more work to find where these errors might be located.

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

$$$$$$$

ORACLE
Academy

JF 3-4
Developing and Testing an Application

20

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