**1.1.4 Documentation & Variables in Java**

**Introduction**

Our tour of Java using Greenfoot continues with a tour of documentation and variables in Java. Variables come in two types: those that contain primitive data and those that contain objects. We will get a bit of initial experience working with both.

Concepts covered are:

* API
* Javadoc
* Comment
* Primitive data types - boolean, byte, short, integer, long, float, double, character
* Object data type
* Dot notation
* Method return types
* Declaration
* Assignment
* Keywords: **boolean, byte, short, int, long, float, double, char, class**

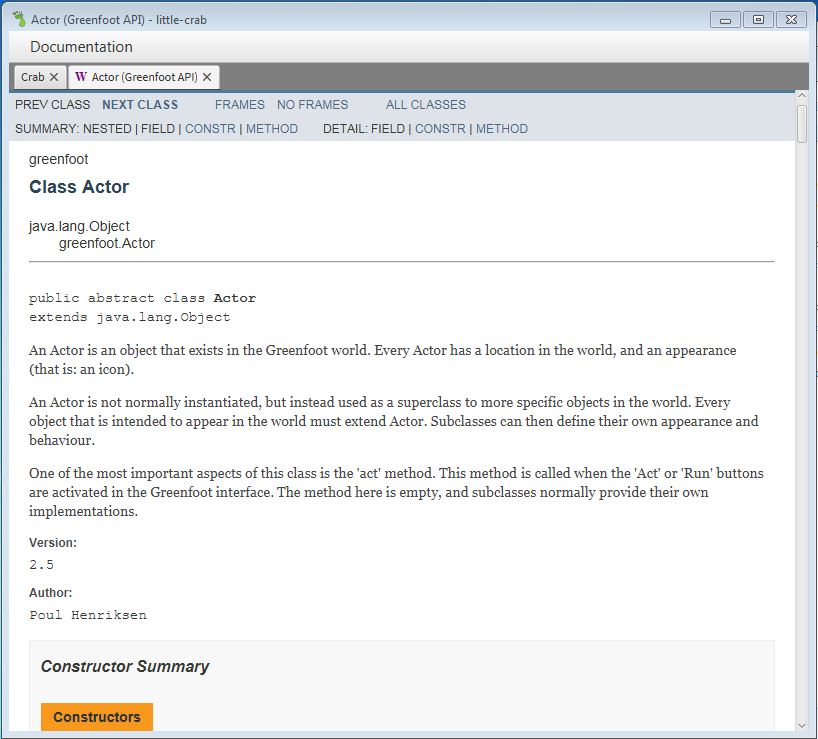
**Materials**

* Computer with Greenfoot
* Source files - little-crab.zip (1.1.4) with completed edits

**Activity**

**Part I: Documentation & Primitive Data Types**

1. Open up your little-crab project from 1.1.4. Today we are going to learn about **documentation** and **variables**.
2. Double-click the Actor class. You will see the **API Javadoc** for the Actor class, which is a Greenfoot built-in class and thus cannot be directly edited. The API tells you what methods exist, what parameters they need, and what they do:



1. Look at the methods. Remember that they have a modifier / type and a description. For example, the method act() has the modifier **void**. The modifier tells you the return value of the method. For example, act does not return a value and thus its modifier is void. Calling this method is like giving a command. In contrast, the method **int** getRotation() has a modifier of integer, which means it gives you back a whole number. Calling this method is like asking a question.

Method return values come in **primitive data types** and **objects**. There are 8 different **primitive data types** that can be returned in Java. Here is a summary list of all **primitive data types**  in Java by keyword:  
  
**boolean** (true or false, 1 bit)

**byte** (number, 8 bits)  
**short** (number, 16 bits)  
**int** (number, 32 bits)  
**long** (number, 64 bits)  
**float** (float number, 32 bits)  
**double** (float number, 64 bits)  
**char** (a Unicode-16 character, 16 bits)

|  |
| --- |
| 1. Write code that tells your crab to turn back if it is touching the edge of the screen. You will need an **if** statement.    * Look through the API and find a method that will help you stop your crab from going off the edges of the screen. If you get an error, make sure your code has no errors. If it does, fix them and compile.    * The method should return a boolean.    * Write an if block in the moveAndTurn() method and place method you found in the round braces of the if block.    * You will need two more methods to make the crab change course when it hits the edge. Type these between the curly braces and don’t forget to use a semicolon to end each line.    * Compile the code, correct any compile-time errors, then create a new crab, and run the scenario. Did it work?    * Add this code to the moveAround() method of the Lobster class as well. |

1. Previously you used the random number generator in Greenfoot. The generator is called getRandomNumber() Remember to use dot notation when using methods from other classes or objects.

|  |
| --- |
| * + Open the Greenfoot API Javadoc by clicking **Help > Greenfoot class documentation**   + Find the description for the getRandomNumber method. Which class is it stored inside?   + See how well you can use the API and dot notation: Modify your lobster code so that you can slow hostile lobsters by clicking on them. Use the API Javadoc for the Greenfoot class to find appropriate methods to accomplish this. To get the object running the code, you can use the **this** keyword. More on **this** later! Place the code in a new method called delayOnClick() and call the method in the act() method. |

1. Every program should include documentation. You have relied on documentation to complete this game and others may rely on yours to write their own programs. To comment in Java, use the // for a single line and /\* to open a multiline comment (and \*/ to close it).

|  |
| --- |
| * + Write comments that explain any methods and code blocks you added in 1.1.3 and 1.1.4. Make the comments in all of your classes.   + Make sure you have a Javadoc header in each of the classes you wrote or modified. Javadoc comments use /\*\* to open and \*/ to close. They appear in blue text in the Greenfoot editor. When you write a header for YOUR Javadoc, you should use Javadoc tags. Add these tags to your header:   @author (the programmer(s) who wrote the code) @version (a version number or date last edited. Updated at the end of every major work session.)   * + Now make Javadoc comments before every method. Add these tags to your Javadoc comments on methods:   @param (the parameters the method needs, if any) @return (the value the method returns, if any)  Every program should include documentation. You have relied on documentation to modify this game and others may rely on yours to write their own programs. To comment in Java, use the // for a single line and /\* to open a multiline comment (and \*/ to close it).  Keep up your Javadocs as we improve the program. |

1. We already have a class for the crab -- we're now going to add another for the worm. Worm is also going to be an Actor.

|  |
| --- |
| * Create a new subclass of Actor * Call it Worm. * From the left-hand list of images, select "worm.png" as the image and press okay. |

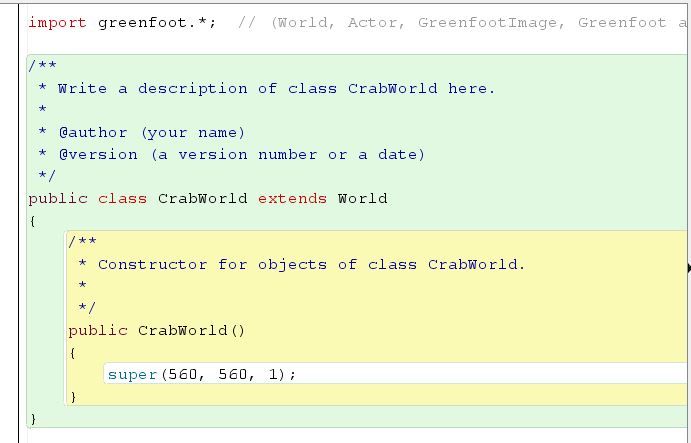
1. Let’s leave the worm code empty for now, and go back to the Crab class. We want the crab to eat the worm. At the end of crab’s act() method, we're going to insert some code to check whether the crab is currently touching a worm. This is branching behavior so we need an if structure and a conditional.

|  |
| --- |
| * Create a new method called eat(). Inside create an if structure. * Use the API for Actor to find a method that returns a boolean and will test to see if one object is touching another. Invoke the method inside the conditional for the **if** structure.   + The method will most likely need an object argument. Since we want to see if we are touching any object of class Worm, the argument would be Worm.**class**   + If you want to remove a specific object, the procedure is more complex. Declare a new variable of the object’s type, assign it using a ‘get’ function of some kind, then remove that particular object using its variable name (see below). * Find a method that will remove any worm object that you touch. Call the method inside of the **if** structure. Compile your crab code. * When you have that done, open the Lobster class. Do the same but make sure he is eating crabs instead of worms! Add this code to his act() method. |

Note how you can use an object as a parameter! Objects are simply complicated data types to Java.

**Part II: Variables**

1. Double-click on the CrabWorld class and take a look at the source code:



What you are seeing is the method that builds CrabWorld. This is called a constructor and will be covered in depth in the next assignment. Suffice it to say, this is what creates a new CrabWorld object. We use the keyword **new** when using the constructor.

The method sets the properties of the instantiated object. For example, this code super(560, 560, 1); sets the dimensions of CrabWorld instances to 560 cells x 560 cells. We are going to use variables to enhance the world.

1. Using variables requires 3 steps:
   * **Declaration**: Creates a new variable with a null value that will contain a specified data type. This usually happens at the beginning of a method or class. A declaration statement has the format type variableName.
     1. Ex: **int** number;
     2. Ex: World world;
   * **Assignment:** Puts a value into the variable. The new value overwrites any previous value. The value must be of the type specified in the declaration.
     1. Uses the assignment operator (**=**)
     2. A method that returns the specified type can be used to assign the variable.
     3. Ex: number = 12;
     4. Ex: world = getWorld();
   * **Use** the variable.

Sometimes we can combine the declaration and the assignment into one statement. Here’s an example:

Crab myCrab = **new** Crab();

Basically this is the equivalent of:

Crab myCrab; (declaration)

myCrab = **new** Crab(); (assignment)

Let’s break this down:

|  |  |  |  |
| --- | --- | --- | --- |
| **Data type** | **Variable name** | **assignment operator** | **variable value or object reference** |
| Crab | myCrab | = | **new** Crab(); |
| int | myNum | = | 52; |
| boolean | myBool | = | true; |
| boolean | anotherBool | = | isTouching(); |
| Lobster | lobster | = | getObject(this) |

|  |
| --- |
| Use the Greenfoot randomizer function you used before to help you add code to the CrabWorld() method of the CrabWorld class that will:   * Create one crab in a *fixed* location * Create one lobster in a *random* location * Create 10 worms in *random* locations. * documentation). You will also have to use constructors and declare new variables.   For now the code will be very redundant - we can fix this later. once we learn to use loops.  Note that the world is 560 x 560, but we start counting at zero. This means the largest coordinate value is 559. The top left of the screen is (0,0) and the y-axis increases values as you go down. This is the same in most programming languages when dealing with a display.  If you need help, look at the API for the World class. |

**Conclusion**

Answer the following questions:

1. What is the syntax for a single-line comment? A multiline comment?
2. What is a class diagram? What information can you get from it? Where is it in the Greenfoot IDE?
3. What is an API? How is it useful? In the Greenfoot IDE, where would you find the Greenfoot API?
4. If I want to know the location of every object around a crab, what method would I use (look in the Greenfoot API)? Write the line of code needed to call the method on object crab of class Crab.
5. What is a Javadoc? What does the @return tag do and where would I place it in the code?
6. What is the difference between a primitive type and an object type? What are the eight primitive types and what do they contain?
7. Combine these lines into one:

int age;

age = 12;

1. What is wrong with the following code and how would you correct it?

int crab = new Crab();

1. Write a variable declaration for isHungry that returns a boolean value. Write a variable declaration that will instantiate an object of class Button.
2. Look at the following code:

int r = 1;

int s = 2;

int t = 3;

r = s;

s = t;

System.out.println(r,s);

What would be printed?