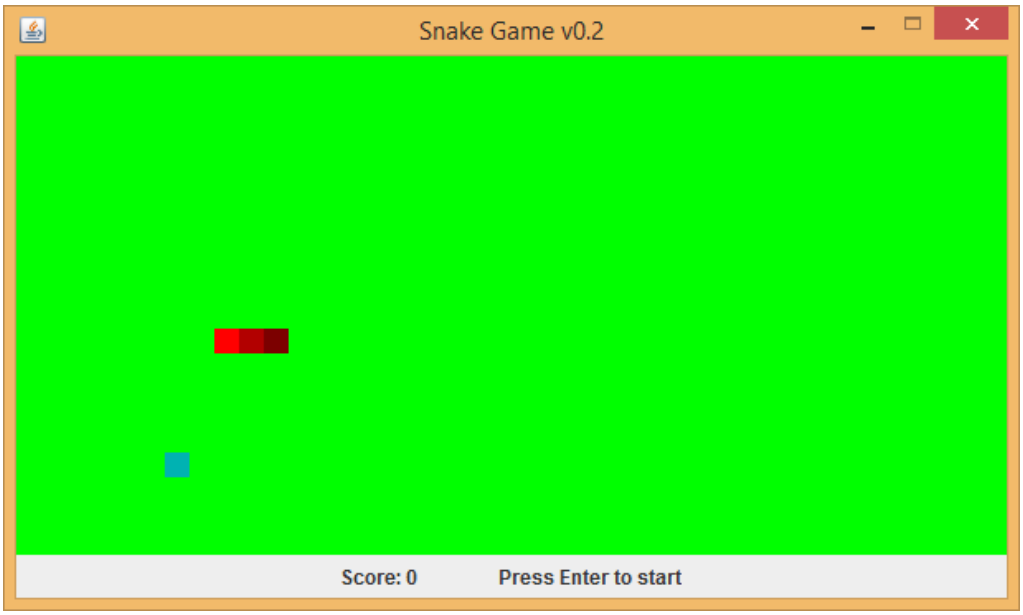
Intro to Java and OO – Snake Game Lesson 1



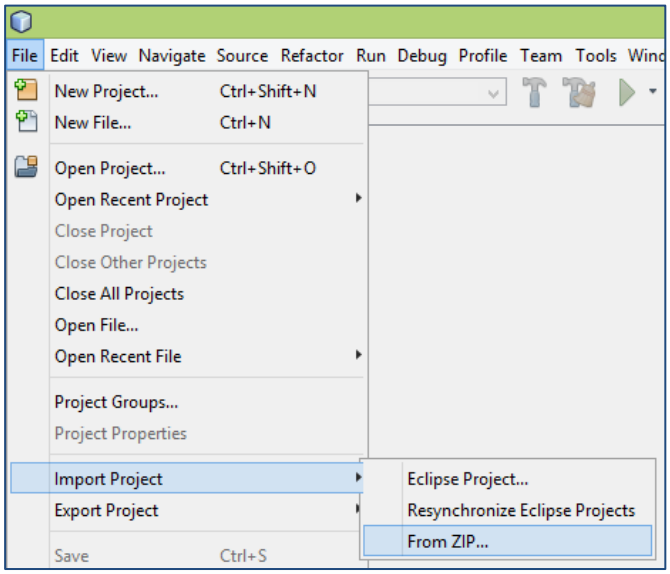
Introduction

Welcome to the introduction to creating a game of Snake in Java!

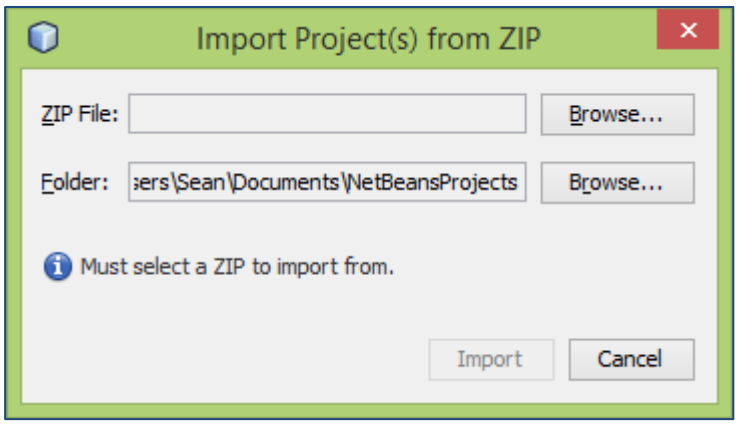
We will be developing the Snake game in a program called **Netbeans**, which is an Integrated Development Environment (IDE) well suited to Java development. It has syntax highlighting (meaning different keywords, and some types of variables are coloured differently in the editor, making the code easier to read), and shows any compiler warnings or errors to help you quickly spot any bugs in your code.

ACTION: OPEN NETBEANS FROM THE START MENU

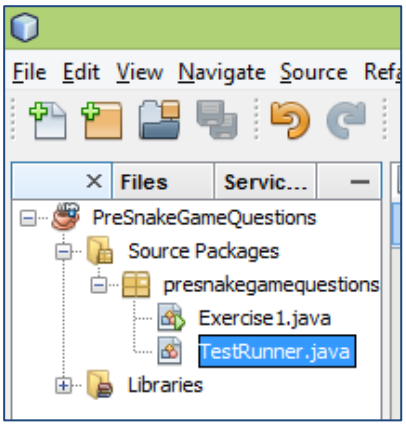
ACTION: When Netbeans opens, Click on File > Import Project > From ZIP…



The location of the zip should be written on the board.

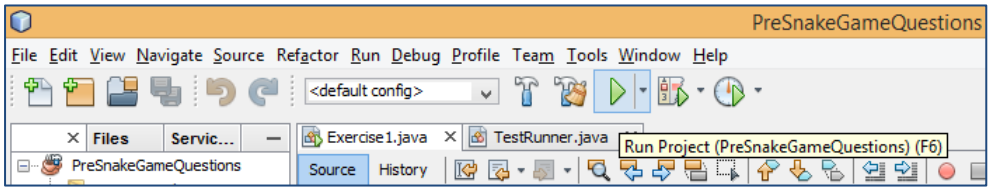


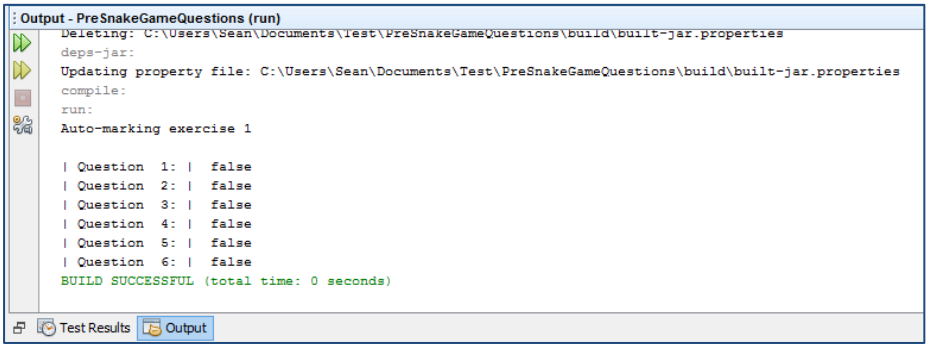
You should see the following once you have imported the project:



Testing Your Code

It's very rare when developing software to write perfect code the first time around. You're encouraged to test early and often, which you can do by clicking on the play button:



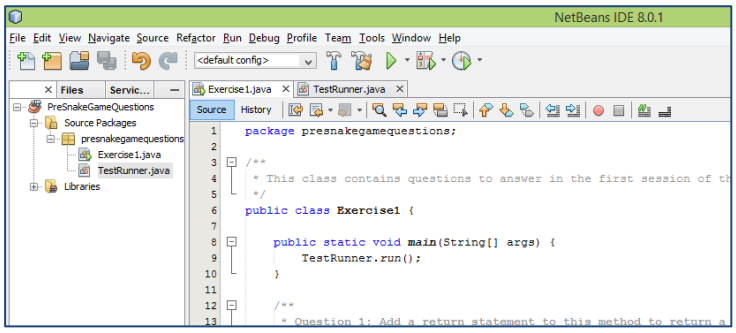


When you run the code, it should come up with an Output pane at the bottom, which shows you how you did. The question was correct if it displays **true** by the question number, and incorrect if it shows **false**. Initially it will only show the answers for the first 6 questions, but as you progress it will automatically mark more questions.

Session One Activities

Action: Open Exercise1.java

Exercise1.java should open in a new editor tab. It contains some skeleton code to provide a structure for your answer to the questions. The actual question is included in the comments above the relevant question method.



Action: Follow the instructions outlined in Exercise1.java to answer the questions. Background information and explanations can be found in this worksheet.

Question One

If a method has a return type (i.e anything but **`void`**), then the method must include a return statement which allows it to return a value of the specified type. Question one asks you to write code to return a double. An example of returning a String is shown below:

```
return "Hello world";
```

Question Two

Question two covers your understanding of data types. Although the question only asks about integers (whole numbers), see if you can figure out what types the other lines would be. You can include your answers in comments after the value:

e.g `number = true; // boolean`

Question Three

So far we've only covered code that runs sequentially , meaning each line is run one after another, but one way to change the flow of the program is using an **if / else statement**. The computer will evaluate a condition put between the brackets of the if statement, as follows:

```
if (<condition>) {
      // Do something
} else {
      // Do something else
}
```

The <condition> is a **Boolean** expression, meaning it can either be true or false. If the condition evaluates to true, then the code inside the if branch is executed, otherwise the code in the else branch is executed.

E.g.

5 > 3 (true)

6 != 6 (false, != means does not equal)

Question Four

This question is very similar to the previous question, except we can also add one or more else if statements. This means you can evaluate multiple different cases before it goes to the else branch. One reason you might want to do this is if you had a String that contained a day of the week, and you could use this kind of statement to test which day of the week was in the String.

Question Five

This question introduces **for loops**. This gives us a way to loop over the code in the brackets a set number of times. The syntax for a for loop is shown below. A for loop has 3 statements within the rounded brackets, which I will explain here:

```
for (int i = 0; i < 10; i++) {
	// Do something
}
```

- `int i = 0`: This is the initialisation statement, which gives the starting point to loop from, in this case 0.
- `i < 10`: This is evaluated every time we execute the code in the loop, and the loop stops when it evaluates to false (i.e when i is greater than or equal to (>=) 10)
- `i++`: This details what happens after each **iteration** around the loop, and i++ just means 'increase i by 1'

Question Six

Now that we have covered for loops, we will now cover **while** and **do-while** loops, which are very similar. A for loop is good for when we know exactly how many times we want to loop over the code, but what if we don't? That is where a while loop comes in. A while loop has the following format:

```
while (condition) {
	// do something that will eventually
	// change condition to be false
}
```

So here we can do whatever we want in between the brackets, but we have to make sure to write code so that the condition will eventually return false… otherwise we get an infinite loop!

The difference between a while loop, and a do-while loop (shown below) is that a while loop will execute **0 or more times**,but the do-while loop will always execute **at least once.** This is because in a do-while loop the condition is evaluated **after** the code has been executed once, so even if the condition starts off false the code will still get executed. **Note the semicolon at the end of the do-while loop!**

```
do {
	// do something
} while (condition);
```

Question Seven

This question introduces the Boolean 'and' operator: **&&**. This operator evaluates the boolean condition on either side of it, and will evaluate to true if and only if **both** sides are true. For example:

(5 > 3) && (2 < 20)	**true**
(10 >= 10) && (3 == 7)	**false**, as the right hand side is false

Question Eight

This question introduces the Boolean 'or' operator: **||**. This operator does the same as the Boolean and operator we saw in the last question and it will evaluate to true if **either** side is true. For example:

(3 < 3) || true	**true**
(2 <= 1000000) || (2 >= 2000)	**true**
20 > 5 || 20 >= 5	**false** – both sides are false

Question Nine

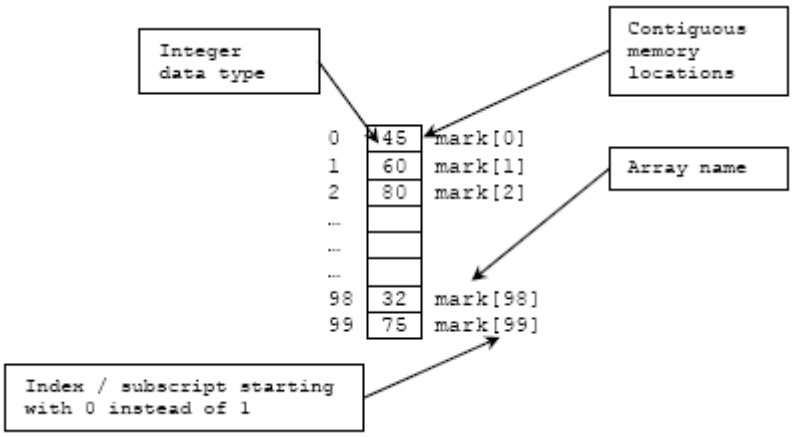
The last type of Boolean operator is the 'not' operator: **!**. This is different to the previous two in that this applies to only one statement, rather than 2. The 'not' operator, as you may be able to guess, inverts the answer, for example:

!true	**false**
!false	**true**
!(5 >= 3)	**false**

Question Ten

Next we will cover **Arrays**. These are a type of data structure that holds a series of objects of the same data type. This means you can have an int array, double array, or any other type you can think of (including any object you create!). This gives us a powerful way to store related data, or data that we want to be able to iterate over.

The diagram below shows how an int array works, don't worry if you don't understand this though, it should become clearer through practice. If not then feel free to ask!



To declare, or **instantiate**, an array, is slightly different to what we've seen before:

```
int[] myArray = new int[10];
```

The square brackets by the type tells us that it is an array, and then when we actually instantiate the array, we have to tell Java how much space to allocate, i.e how many items the array will contain. In this case it's 10. We also have to use the **new** keyword, as an array is a type of object, different to types such as int, double, char, etc. that are known as **primitives.** We'll cover this more in the next lesson.

To store something in an array, or to read a value in the array, we have to have a way of accessing the elements inside the array. We can do this as follows:

```
myArray[0] = 20;
myArray[1] = 15;
…
myArray[9] = 3;
```

IMPORTANT: Arrays are zero-indexed, meaning that when you access the first element, you access it using 0, as shown on the left. This also means that the tenth element is accessed using index 9.

Question Eleven

This question aims to bring together two of the things you've just learned about: for loops and arrays. In this question you are asked to create an int array that can hold 10 items, and fill the array with values 0 to 9 using a for loop.

Hint: Since the value in the array will be the same as the index for that place in the array, we can use the 'I' variable from the for loop as the value. For example:

```
myArray[i] = i;
```

Question Twelve

Next you are asked to print something to the console (output window)! This is a very useful statement, allowing you to print out information to help you diagnose and find out what might have gone wrong with your code! The most common way of printing out something to the console is by using the following:

```
System.out.println("Your message goes here!");
```

You can pass almost any type of object to this method, and it will convert it to a String and print it out for you!

Question Thirteen

As Java doesn't take whitespace (e.g tabs, spaces, etc.) into account when compiling the code, it doesn't strictly have to be formatted perfectly like languages such as Python. However it is best practice to always try and write code that can be easily readable, and therefore easy to change in the future!

As you can see, all the code in this question is valid (there are no compile errors), but it's up to you to format it so it looks nice and readable! Use the following as a guideline:

- One statement per line (i.e no `String odds = ""; String evens = "";`)
- Proper indentation of anything in between curly brackets (usually a tab or 3 spaces)
- This includes putting code like '`evens += i;`' on a new line!
- Spaces in between the operators such as equals, and inside the different parts of the for loop

Question Fourteen

This question will be more difficult than the previous ones, as you are asked to make your own design decisions, so don't be afraid to ask for help if you're stuck! One of the best practices in Software Development is code **reuse**, meaning that we can reuse bits of code easily, without having to rewrite very similar code multiple times.

After looking at the code in question 14, hopefully you can see that the blocks of code (with a blank line between them) are almost identical! The only real difference between them is the variable that contains the result, and the starting number.

Wouldn't it be great if we could just write this code once, giving a starting number **parameter**, and getting the correct result returned to us? Luckily, this is not only possible, it is good practice! We can take the repeated code and put it in a separate method, and then call that instead of writing the code out again!

Hint: The method you create should take an int parameter, telling it the number to start off with, and it should return an int value, after the mathematical operations have been performed.