CSY2030
Systems Design & Development
Revision of Java 3

Overview of Lecture

- Today we will revise the following in Java:
  - Arguments for Methods
  - Value Returning Methods
  - User Input
  - String Manipulation
  - Random Numbers

Arguments

- When you declare a method it's followed by an opening an closing bracket:

```
public static void myMethod ( ) {
}
```

- These brackets can contain one or more “arguments”
- An argument is a variable which will then be available in the method when it is called
- When the method is called, the value is set

Arguments

```
public  class VoidMethodExample{

public  static void main(String[] args) {
        myMethod(1);
        myMethod(78);
}

public static void myMethod(int myArgument) {
        System.out.println(myArgument);
}
}
```

```
1
78
```

Arguments

- To call a method that has an argument, use the method name followed by the value in brackets e.g *myMethod(123);*
- If a method is declared with an argument, an argument must be provided when it is called.
- The value sent to the method when it is called must be the same type as the type asked for in the method definition

Arguments

```
public class VoidMethodExample {

public static void main(String args[]) {
    myMethod();
    myMethod("Argument");
    myMethod(134);
}

public static void myMethod(int myArgument) {
    System.out.println(myArgument);
}

}
```

Invalid: argument not supplied

Invalid: argument is wrong type

Valid argument is supplied and right type

Arguments

- You can declare multiple arguments by separating them with a comma.
- You must also supply the value for the arguments when the method is called

Arguments

```
public class VoidMethodExample {

public static void main(String[] args) {
        myMethod(1, “Jim”);
        myMethod(78, “Fred”);
}

public static void myMethod(int firstArgument,
                String secondArgument) {
        System.out.println("First argument is " +
                firstArgument +" Second argument is " +
                secondArgument);
}
}
```

```
First argument is 1 Second argument is Jim
First argument is 78 Second argument is Fred
```

Arguments

- When calling a method with more than one argument the order matters.
  - The order when called must match the order in the method definition
- The following won't compile because order is wrong:

```
public class VoidMethodExample {

public static void main(String[] args) {
       myMethod(“Jim”, 1);
}

public static void myMethod(int firstArgument,
                String secondArgument) {
       System.out.println("First argument is " +
                firstArgument +"Second argument is " +
                        secondArgument);
}
}
```

Void methods

- Each method you have written so far uses the code

```
public static void main(String[] args) {
}
```

- This is a “void” method
- Void methods run the code in order but do not return a value (produce a result)

Value returning methods

- Methods are either “void” or value returning
- Value returning methods can return a value that can be used in the calling method
- Value returning methods are declared with a type (e.g. *int*, *double*, *String*) in place of *void* in the method header. A method that returns an *int* would look like this:

```
public  static int   myMethod()  {

}
```

Value returning methods

- Once a method is declared as a value-returning method it must also include the keyword *return*
- The return keyword is followed by the value that is "returned"
  - The return must be the last statement in the method e.g

```
public static int myMethod(){
    return 45;
}
```

Value returning methods

- You can combine arguments with return values to create a method that performs a task

```
public static void main(String[] args)
{
    int result = add(4, 5);
    System.out.println(result);
}
public static int add(int num1, int num2) {
    int result;
    result = num1 + num2;
    return result;
}
```

```
9
```

Value returning methods for Validation

- Returning a boolean value can be useful for validation
- You can call a method directly inside an if statement

Value returning methods for Validation

```
public class ValidationExample {

    public static void main(String[]  args) {
        int num;
        num = 12;
        if (isValid(12)) {
            System.out.println(num + " is valid");
        }
        else {
            System.out.println(num + " is not valid");
        }
    }

    public static boolean isValid ( int number)
    {   if (number <= 100 && number > 0) {
            return  true;
        }
        else {
            return false;
        }
     }
}
```

User Input

- Getting input from a user allows the person using the program to change the result. For example entering two numbers to be calculated
- User input uses the inbuilt Java *Scanner* class

User Input

```
import java.util.Scanner;

public class ScannerExample {
    public static void main(String[] args) {

        Scanner myScanner = new Scanner(System.in);

        System.out.println("Enter a value");

        String userInput = myScanner.nextLine();

        System.out.println("You entered: " + userInput);


    }
}
```

1. Import scanner class

2. Create a scanner object

3. Capture the user input and store it in a String variable

4. The user input is now usable like any other variable

String Comparison

- When comparing primitive types (*int*, *float*, *double*, *byte*, *char*, *long*, *boolean*) you can use the equality (==) operator e.g:

```
int num = 4;
//Integer Comparison
if (num == 4) {}

boolean boolValue = false;
//Boolean comparison
if (boolValue == true) {}

double doubleValue = 32.43;
//Double comparison
if (doubleValue == 53.45) {}
```

Do not use = for comparison – this is used for assignment

String comparison

- When comparing Strings you must use the *equals()* method on the string itself because == will not work
- Code would look something like:

```
Scanner myScanner = new Scanner(System.in);
System.out.println("Enter a string:");
String myString = myScanner.nextLine();
if (myString.equals("Hello")){
   System.out.println("Hello entered");
}
else {
   System.out.println("Hello not entered");
}
```

String manipulation

- You can use the *string.length()* and *string.charAt(i)* methods with a *for* loop to show a character at a specific position in the String

```
String myString = "Hello";
for (int i = 0; i < myString.length(); i++) {
    char chr = myString.charAt(i);
    System.out.println("Character at position " + i + " is " + chr);
}
```

```
Character at position 0 is H
Character at position 1 is e
Character at position 2 is l
Character at position 3 is l
Character at position 4 is o
```

Finding out if a string contains a specific character

- You can use *if* statements and a *for* loop to find out if a String contains a specific character
- E.g. to find out of a string contains the letter “D”

```
String searchString = "Systems Design & Development";
for (int i = 0; i < searchString.length(); i++) {
      if (searchString.charAt(i) == 'D') {
            System.out.println("Letter D found");
      }
}
```

Random numbers

- Java has an inbuilt mechanism for generating random numbers using the *Math.random()* method

```
double random = Math.random();
System.out.println(random);
```

Random numbers

- If you want a random number between 1 and 100, add 1 to the result:

```
int random = (int) (Math.random()*100)+1;

System.out.println(random);
```