Operators

Comments in Java

To make the code more readable, we can include comments/notes in a class; it can be literally anywhere in the class. But ideally we will include it in places where it would make most sense. A one line comment is included with a prefix '//'. A multi-line comment is included with a prefix of '/*' *and suffix of* '*/'.

```
//This is the first class we created in Java
public class HelloWorld {
/*This is the only method in this class.
This method prints Hello World! on the console.*/

  public static void main(String[] args) {
    System.out.println("Hello world!");
  }
}
```

The compiler ignores the comments and only compiles the actual code and converts it to bytecodes. Try to make changes to the original file you wrote and see if it makes any difference when you run the java application.When comments are written in a particular format, these will help generate documentation for the class files. That is beyond the scope of this session.

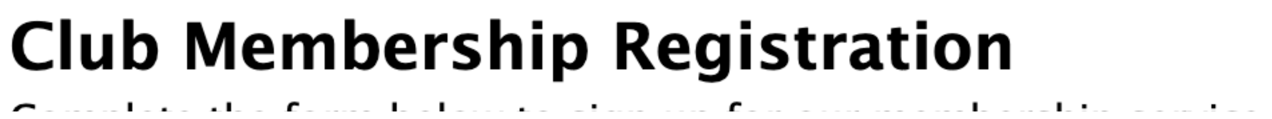
Java is an Object Oriented Programming Language. Object Oriented Programming as the name says is based on Objects. What are objects? Objects are instances of a class. Any class in Java can have one or more of the following three purposes.

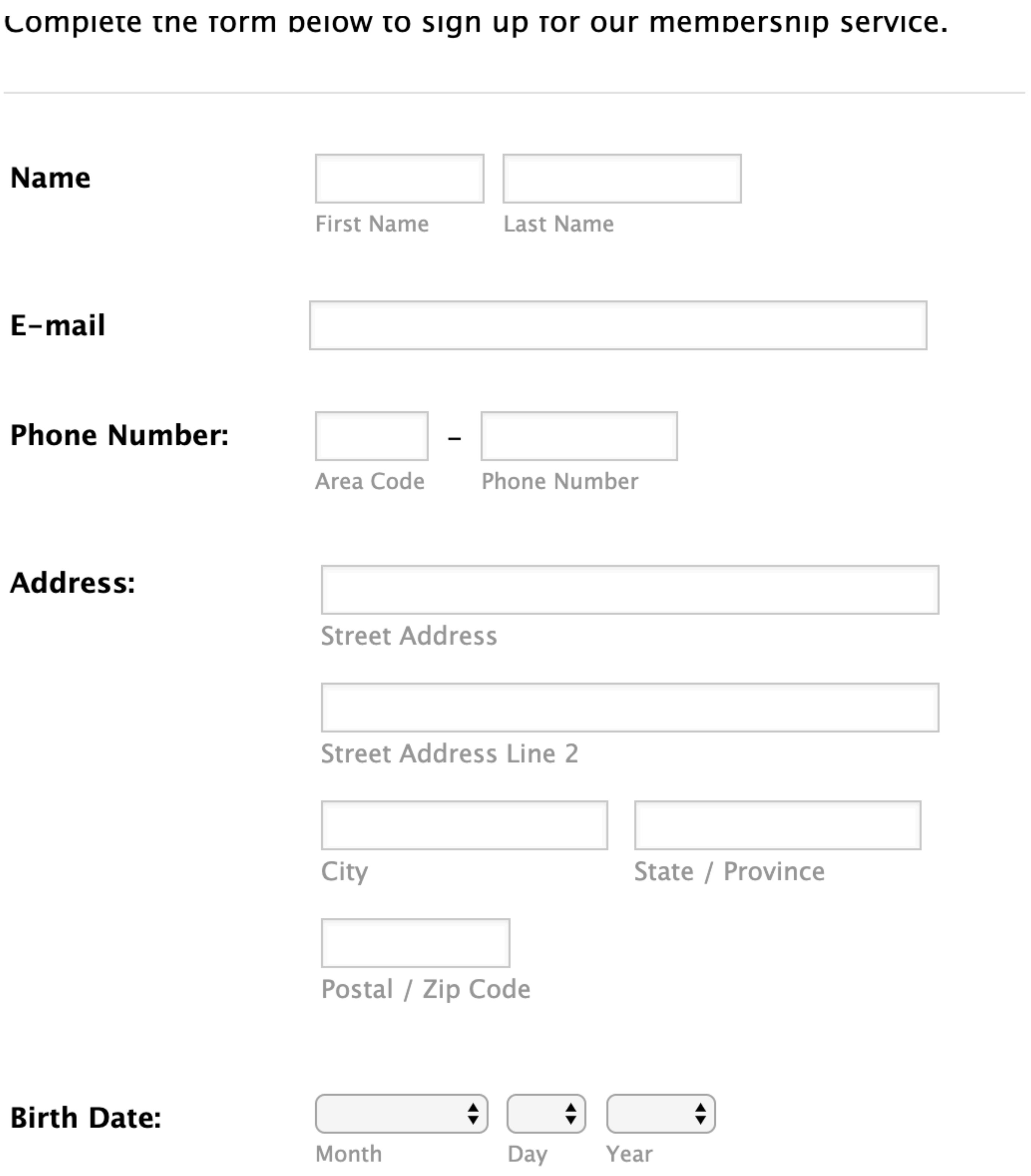
1. Runnable class
2. Use through instantiation
3. Use through inheritance

When a JVM runs, an object is not constructed in the memory until it needs to be. When we ran the Runnable class, it only loaded the class and any memory required to print the String out was used. When is the object constructed in that case? How do we create objects? What exactly happens when you construct the object? Before we answer these questions, we should understand what a class is.

Classes are more like reusable templates. Look at the images below.

Class


Club Membership Registration


Complete the form below to sign up for our membership service.

Name
First Name
Last Name

E-mail

Phone Number:
Area Code – Phone Number

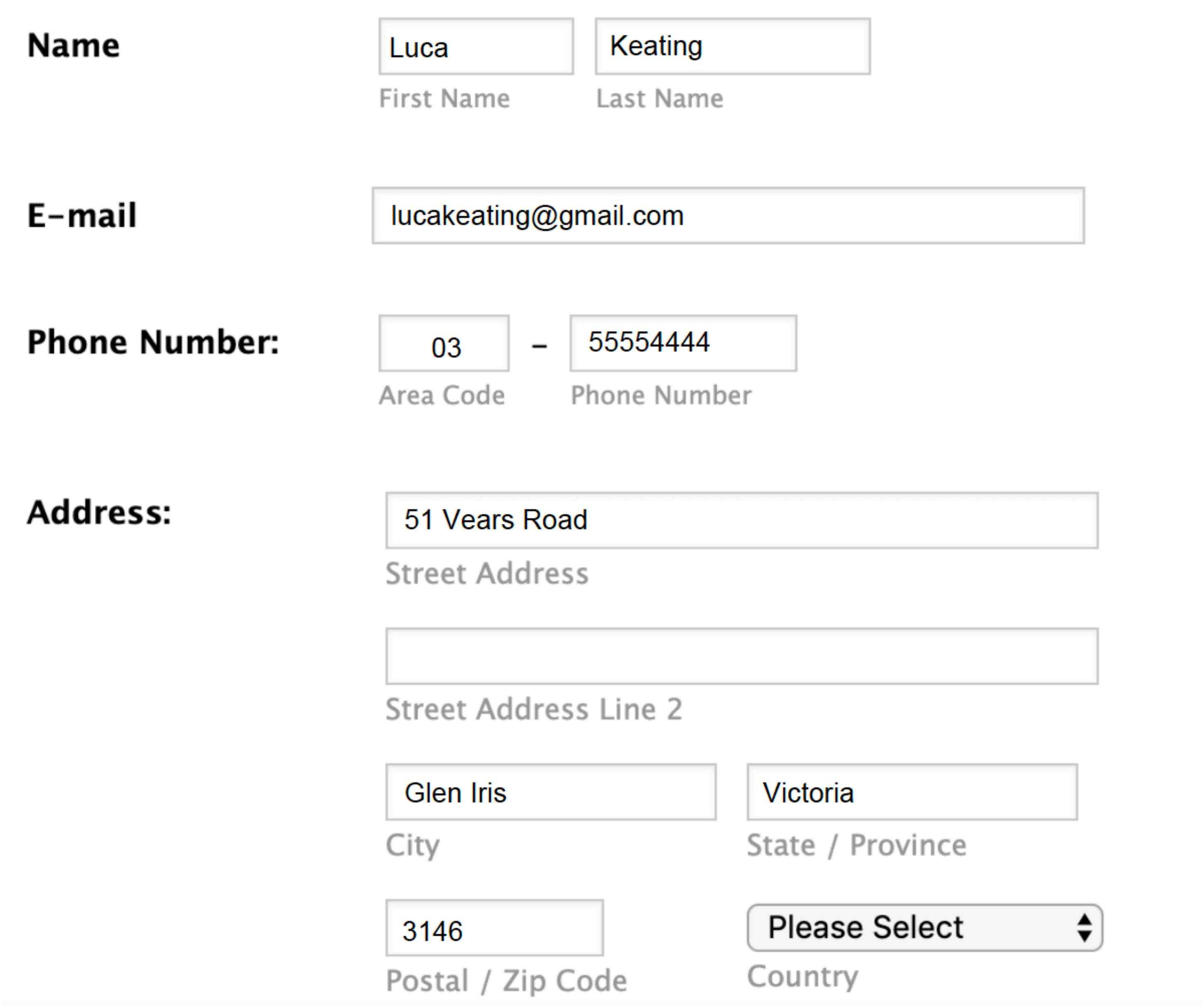
Address:
Street Address
Street Address Line 2
City
State / Province
Postal / Zip Code

Birth Date:
Month
Day
Year

Object

Club Membership Registration

Complete the form below to sign up for our membership service.



Name	Luca (First Name)	Keating (Last Name)
E-mail	lucakeating@gmail.com	
Phone Number:	03 (Area Code)	55554444 (Phone Number)
Address:	51 Vears Road (Street Address)	
	(Street Address Line 2)	
	Glen Iris (City)	Victoria (State / Province)
	3146 (Postal / Zip Code)	Please Select (Country)

Members of a class

A class can have attributes, methods and special methods called constructors.

Attributes are distinct features of the class. Take the case of a bank customer. What are the attributes you can think of?

1. First Name
2. Second Name
3. Email
4. Account number

5. Type of account
6. Address for communication

Let's now put this in a class, Customer.

```
public class Customer {
  String firstName;
  String secondName;
  //Try to finish the rest of it
}
```

Methods in Java, like in every other language, can have parameters. But a method can only exist inside of a class. And every method must have a return type. The methods can be public, protected, private or default (default is when you specify nothing; don't actually write the word default). In the runnable class we saw the method `public static void main(String s[])`. This is a public method which takes a String array as argument returns nothing, hence `void`. Static is a non-access modifier. We will learn about this later.

The attributes can have different kind of access. They can be public, protected, private or default (default is when you specify nothing; don't actually write the word default). These access modifiers are a vital part of encapsulation, one of the most important features of Object Oriented programming we will be learning about.

Constructors are methods which have the same name as the class. As they construct and return the object, they don't have a return type. They can also have public, protected, private or default access.

To create an object of a class, we use the `new` keyword. Let's create an object of the customer class we created earlier, within our runnable class.

```
public class MyRunnableClass {
  Customer customer = new Customer();
  System.out.println(customer);
}
```

We never wrote any constructor for the class. How did that even work? This is because, the default constructor, the constructor which takes no arguments, is implicitly created for the classes by Java. However, if we have something specific to be done while the object is contructed, we can explicitly define it. Add these following lines to your Customer class.

```
public Customer() {
  System.out.println("Customer object created");
}
```

Activity - Now run the runnable class and see the constructor being called.

Wrapper Classes

We dealt with many primitive datatypes yesterday. The disadvantage with the primitive datatypes is that while they can be used for operation you cannot operate on them. For instance if we had a String which in primitive datatype is a char[] and we wanted to get a substring of that, how would we do that? It is a very tedious process to treat the array and extract elements. This is where the wrapper classes come into picture. Each of the primitive datatype has a Wrapper class.

char	Character
char[]	String
int	Integer
byte	Byte
short	Short
float	Float
double	Double
boolean	Boolean

Activity: Try creating each variable you created with primitive data types earlier, with the wrapper classes.

```
Byte b =new Byte("127")

System.out.println(b)

Short sh = new Short("32767")

System.out.println(sh)

Integer j = new Integer("2147483647")

System.out.println(j)

Long l = new Long("9223372036854775807")

System.out.println(l)

Float f = new Float("99999.0")

System.out.println(f)

Double d = new Double("9999999.999999999")

System.out.println(d)
```

Activity :

Find the max and the min value of each data type and print it out.

String class and some useful functions:

When we create a String object in Java using `String s = "Java is cool";` what we are actually doing is `String s = new String("Java is cool")`. This is done implicitly for all most commonly used Java classes like String, Integer, etc.,

String class provides many useful functions. We will look at a few here. You can refer the **documentation (https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/String.html)** for more. You can try the following examples in JShell or within the runnable class you created.

```
        String s = "Java is cool!";

        System.out.println("Char at "+s.charAt(0));
        System.out.println("Compare "+s.compareTo("Java is hot!"));
        System.out.println("Concat "+s.concat("So am I."));
        System.out.println("Contains 'cool' "+s.contains("cool"));
        System.out.println("Contains 'hot' "+s.contains("cool"));
        System.out.println("Ends with 'cool!' "+s.endsWith("cool!"));
        System.out.println("Ends with 'hot!' "+s.endsWith("hot!"));
        System.out.println("Index of 'cool'"+s.indexOf("cool"));
        System.out.println("Index of 'hot'"+s.indexOf("cool"));
        System.out.println("Index of space"+s.indexOf(" "));
        System.out.println("Last index of space"+s.lastIndexOf(" "));
        System.out.println(s.split(" "));

        String sWithSpace = "    Java is so cool!     ";
        System.out.println(sWithSpace);
        System.out.println(sWithSpace.trim());
        System.out.println(s.toLowerCase());
        System.out.println(s.toUpperCase());


        System.out.println("Substring from 5th position - "+s.substring(5));
        System.out.println("Substring from 5th position to 10th position - "+s.substring(5,10));

        String str1 = "Hello";
        String str2 = "Hello";

        System.out.println(str1.equals(str2));
        System.out.println("The equality of the two objects" + str1 == str2);
```

💡 Did you notice that `str1.equals(str2)` in the code above returned true whereas `str1 == str2` returned false. Why do you think this happened?

Operators

When we did `int i=5` we used what is called an assignment operator. We are assigning a value to a variable. `=` is the simplest of all the operators. There are many other assignment operators. Before we look at them let's look at the arithmetic operators.

+	Addition `int i = 4;` `int j = 5;` `System.out.println(i+j);` `+ can also be used with String to concatenate two Strings.`
-	Subtraction `int i = 6;` `int j = 5;` `System.out.println(i-j);`
*	Multiplication `int i = 4;` `int j = 5;` `System.out.println(i*j);`
/	Division `int i = 20;` `int j = 5;` `System.out.println(i/j);`
%	Modulus `int i = 23;` `int j = 5;` `System.out.println(i%j);`
++	Increment by 1Beside `int i = 23;` `System.out.println(++i); //Pre-increment` `System.out.println(i++); //Post-increment`
--	Decrement by 1 `int i = 23;` `System.out.println(--i); //Pre-increment` `System.out.println(i--); //Post-increment`

Besides `=`, the other assignment operators are:

+=	The new value of the variable is the original value plus the operand on the right.
	`int i = 5;`
	`i+=6;`
	`//The value of i will now be 11`

-=	The new value of the variable is the original value less the operand on the right.
	`int i = 15;`
	`i-=6;`
	`//The value of i will now be 9`

*=	The new value of the variable is the original value times the operand on the right.
	`int i = 5;`
	`i*=6;`
	`//The value of i will now be 30`

/=	The new value of the variable is the original value divided by the operand on the right.
	`int i = 18;`
	`i/=6;`
	`//The value of i will now be 3`

Relational operators or Compartitive operators are the other kind of operators. These are pretty similar to other languages. But it is goos to bear in mind that `==` and `!=` compare the objects and not the value of the objects. To compare the value of the objects we will use the `.equals()` method.

==	See if the objects in the left and right are equal. For primitive types, it compares values. int i = 5; int j = 5; System.out.println("i == j "+i == j);

!=	See if the objects in the left and right are inequal. For primitive types, it compares values.
	`int i = 5; int j = 5;`
	
	`System.out.println("i != j "+i != j);`
	
	`System.out.println("i != 6 "+i != 6);`

>	See if the value in the left is greater than the one on the right. `int i = 7;`

	`int j = 6;` `System.out.println("i>j "+i > j);`
<	See if the value in the left is less than the one on the right. `int i = 7;` `int j = 6;` `System.out.println("i <j " + i < j);`
>=	See if the value in the left is greater than or equal to the one on the right. `int i = 7;` `int j = 6;` `System.out.println("i>=j "+i >= j);`
<=	See if the value in the left is less than or equal to the one on the right. `int i = 7;` `int j = 6;` `System.out.println("i<=j "+i <= j);`

Logical operators

||	Or - Used to check either-or conditions.
&&	And - Used to check this and that conditions.
!	Not - Used to check if something is `not` true.

Obtaining input from user

We saw yesterday that to print out anything from a Java program we use System.out which represents the standard output, the monitor. To obtain any input from the user we will use the standard input which is the keyboard. But the input from the keyboard can only be obtained as bytes. To obtain input as String we use a class called Scanner. The Scanner object's constructor takes an input stream as argument. Since we are going to provide input through the keyboard, the argument will be `System.in` which represents the standard input. Scanner is not available in the default packages that are available for java coding. We have to explicitly import the package.

```
import java.util.Scanner;

public class UserInput {
    public static void main(String s[]) {
        //We create a scanner object which will read from keyboard and give it a reference name mySc
anner
        Scanner myScanner = new Scanner(System.in);
        System.out.println("What is your name? ");
        //Read the next line that the user inputs
        String name = myScanner.nextLine();
        System.out.println("Hello "+name);
    }
}
```