Lesson 2.2

Class and Object

By the end of this lesson you will learn:

- Introduction to Class
- Introduction to Object
- Defining A Class
- Defining Instance Variables
- Defining Instance Methods
- Instance Methods Visibility
- Instance Methods body
- Returning value from a method
- Using class as a type
- Initializing object reference
- Invoking instance method
- Constructor in a java class
- Converting a c++ code to java

Class: Classes are templates or blueprints for objects. It defines the methods and data. For example, close your eyes and think about a chair. In your imagination the chair has a seat length, seat width, color etc. But can you touch your chair? No. So, we can call this a class which is the blueprint of an Object.

Object: An object is the appearance of a class. In another word it is the instance of a class. Again, think about the chair you are sitting. It has seat length, seat width, color etc. This time you can touch your chair. So, this is an object.

The process of creating an object is known as instantiation. The attributes of an object are called instance variables. You can create an object using the following syntax.

Syntax to create an object:

```
Employee anEmployee = new Employee();
```

Here, anEmployee is the object of Employee class. New is keyword for memory allocation. This keyword is necessary for all non-primitive data type.

Class definition must have the following :

- The keyword "**class**" followed by the **name of the class**
- The class body

Before the keyword class you can use access modifier “public” which is optional. You can create a class using the following syntax:

Syntax to create a class:

public class Employee

{

//body of the class

}

If a class is public, it must be defined within a file which is the same name as the class with a ".java" extension. For example for the above Employee class the file name must be “Employee.java”

The class body can contain

- Zero or more instance variables
- Zero or more methods

Instance variable: Instance variables are declared using the same syntax as ordinary variables. Variables can be prefixed with a visibility modifier.

Variables can have one of 4 different visibilities:

- **public** - the variable can be directly accessed from anywhere
- **private** - the variable can only be directly accessed from within the class
- **protected** - the variable can be access directly from within the class, within the package, or from within any subclass.
- **default** (no modifier specified) - the variable can be accessed directly from within the package

// We will discuss in details later.

You can create an instance variable using the following syntax.

Syntax to create instance variable:

public String employee_name;

Instance method: Method definitions include a method signature and a method body. Methods signatures are defined with the following syntax.

Syntax to create an instance method:

access modifier return_type method_name(type name, ...)

```
public void setEname(String en)
{
        employee_name=en;
}
```

The return type can be:

- a fundamental data type
- an object reference
- void (no return)

Parameters are optional

- If the method takes no parameters, empty brackets are required ()
- Multiple parameters are separated by commas
- Parameters are defined by type and name
    - A parameter is a local variable whose scope is the method.

Methods have the same visibility modifier as instance variable.

Returning value from a method:

A method which has a non-void return type MUST return a value

- The return value's type must match the type defined in the method's signature.
- A void method can use a return statement (with no return value) to exit the method.
- The return value can be used the same as any other expression.

```
public class Car
{
        private int currentGear;
```

```
        private int currentRpms;
        public int calculateSpeed()
        {
                return currentRpms * currentGear;
        }
}
```

You can use a variable of type “class”. You can declare a variable of class using the following syntax:

Employee anEmployee;

Invoking Instance Method: To invoke a method on an object, use the . (dot) operator If there is a return value, it can be used as an expression**.**

Syntax to invoke a method:

```
        Car aCar = new Car();


        [...]


                if (aCar.calculateSpeed()>110)
                {
                        System.out.println("You're Speeding!");
                }


        [...]
```

Here,” aCar” object is invoking method “calculateSpeed()”.

Passing Parameter to Method: When a method invocation is made, any parameters included in the invocation are passed to the method. Parameters become available within the method. They are not known outside the method.

- All parameters are passed by value. i.e, a copy is made
  - The value of fundamental data types are copied
  - The value of object references (i.e, memory addresses) are copied

```
public float calculateInterestForMonth(float rate)
{
        return lowBalanceForMonth * (rate/12.0);
}
```

“rate” is the parameter in “calculateInterestForMonth()” method.

Initializing Object: When an object is created, all instance variables are initialized to the default value for their type

- Fundamentals are 0, 0.0, '\000' or false
- Object references are null

In order to put the object into a usable state, its instance variables should be initialized to usable values. This could be accomplished by calling the various set methods. This is not always possible because it is not required that all instance variables have set methods.

Java provides for another method of initializing objects. When an object is created, a constructor is invoked. The responsibility of the constructor method is to initialize the object into a usable state.

Constructor: Constructors have the following characteristics

- There is NO return type. NOT even void
- The method name is the same name as the class
- Constructors can be overloaded

Syntax for creating a Constructor:

```
public class BankAccount
{
        String ownersName;
        int accountNumber;
        float balance;

        public BankAccount()
        {
        }
```

```
    public BankAccount(int anAccountNumber)
    {
        accountNumber = anAccountNumber;
    }

    public BankAccount(int anAccountNumber, String aName)
    {
        accountNumber = anAccountNumber;
        ownersName = aName;
    }
    [...]
}
```

Here, In the above example there are three constructor "BankAccount" with different parameter list.

When an object is created (using new) the compiler determines which constructor is to be invoked by the parameters passed. Multiple constructors allows the class programmer to define many different ways of creating an object.

```
public static void main(String[] args)
{
    BankAccount anAccount = new BankAccount();
    BankAccount anotherAccount = new BankAccount(12345);
    BankAccount myAccount = new BankAccount(33423, "Craig");

}
```

- ✓ Here in the 1st line empty constructor is called.
- ✓ In the second line the constructor with one integer parameter [*BankAccount(int anAccountNumber)*] is called
- ✓ In the last line the constructor with one integer and one String parameter [*BankAccount(int anAccountNumber, String aName)*] is called.

If no constructors are defined for a class, the compiler automatically generates a default, no argument constructor. All instance variables are initialized to default values.

However, if any constructor is defined which takes parameters, the compiler will NOT generate the default, no argument constructor. If you still need one, you have to explicitly define one.

A side by side comparison between a C++ and java code:

C++ code	Java Code

#include <iostream>
using namespace std;
 	
class Person
{
 private:
  int id;
  string name;
 	
 public:
  Person()
  {
 	
  }
  Person(int i,string n)
  {
  id=i;
  name=n;
  }
 	
  void displayInfo()
  {
 cout<<"Name is :"<<name<<endl;
 cout<<"Id is :"<<id<<endl;
 	
  }
};
 	
int main()
{
  Person p1(101,"Mashrafi");
  p1.displayInfo();
 	
  return 0;
}
	
	public class Person
	{
	 private int id;
	 private String name;
	
	public Person()
	  {
	
	  }
	
	public Person(int i,String n)
	  {
	  id=i;
	  name=n;
	  }
	
	public void displayInfo()
	 {
	System.out.println("Name is :"+name);
	System.out.println("Id is :"+id);
	 }
	
	public static void main(String args[])
	  {
	Person p1=new Person(101,"Mashrafi");
	  p1.displayInfo();
	  }
	
	}

Exercise:

1. A. Write a class Student that has the following attributes:
   - int id
   - String name
   - double cgpa

   There is a set method and a get method for each of the attributes. There is also a main method in this class. Inside the main method create one object of the Student class and demonstrate all the methods.

2. Look at the following class notation carefully and develop the class.

Class Name	Account
Attributes	int accountNumber String accountHolderName double balance
Methods	void setAccountNumber(int an) void setAccountHolderName(String ahn) void setBalance(double b) int getAccountNumber( ) String getAccountHolderName( ) double getBalance( ) void showDetails( )