Human-Computer Interaction

Practical session

Overview of OOP and standard Java

Java language

- JVM (Java Virtual Machine)
- JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

- When we consider a Java program, it can be defined as a collection of objects that communicate via invoking each other's methods.

Java is an Object-Oriented Language.

- As a language that has the Object-Oriented feature, Java supports the following fundamental concepts:
- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Instance
- Method
- Message Passing

- **Object**: Objects have states and behaviours. Example: A dog has states - colour, name, breed as well as behaviour such as wagging their tail, barking, eating. An object is an instance of a class.
- **Class**: A class can be defined as a template/blueprint that describes the behaviour/state that the object of its type supports.
- **Methods:** A method is basically a behaviour. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables:** Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

Basic Syntax

About Java programs, it is very important to keep in mind the following points.

- **Case Sensitivity:** Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
- **Class Names**: For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.
- **Example:** *class MyFirstJavaClass*
- **Method Names**: All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.
- **Example:** *public void myMethodName()*

- **Program File Name**: Name of the program file should exactly match the class name.
- When saving the file, you should save it using the class name (Remember Java is case sensitive) (if the file name and the class name do not match, your program will not compile).
- **Example:** Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*

- **public static void main(String args[])**: Java program processing starts from the main() method which is a mandatory part of every Java program.

Example

```
public class MyFirstJavaProgram {
    /* This is my first java program.
     * This will print 'Hello World' as the output
     */
    public static void main (String []args){
        System.out.println ("Hello World"); // prints Hello World
    }

}
```

Java Modifiers

- Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There are two categories of modifiers −
- **Access Modifiers**: default, public , protected, private
- **Non-access Modifiers**: final, abstract, strictfp

Access Control Modifiers

- Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are:

- Visible to the package, the default. No modifiers are needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected).

Non-Access Modifiers

Java provides a number of non-access modifiers to achieve many other functionality.

- The *static* modifier for creating class methods and variables.
- The *final* modifier for finalizing the implementations of classes, methods, and variables.
- The *abstract* modifier for creating abstract classes and methods.
- The *synchronized* and *volatile* modifiers, which are used for threads.
- Strictfp: When we declare an interface or a class with strictfp, all of its member methods and other nested types inherit its behaviour.

Example

```
public class className {
// ... }
 private boolean myFlag;
static final double weeks = 9.5;
protected static final int BOXWIDTH = 42;
public static void main(String[] arguments)
{ // body of method }
```

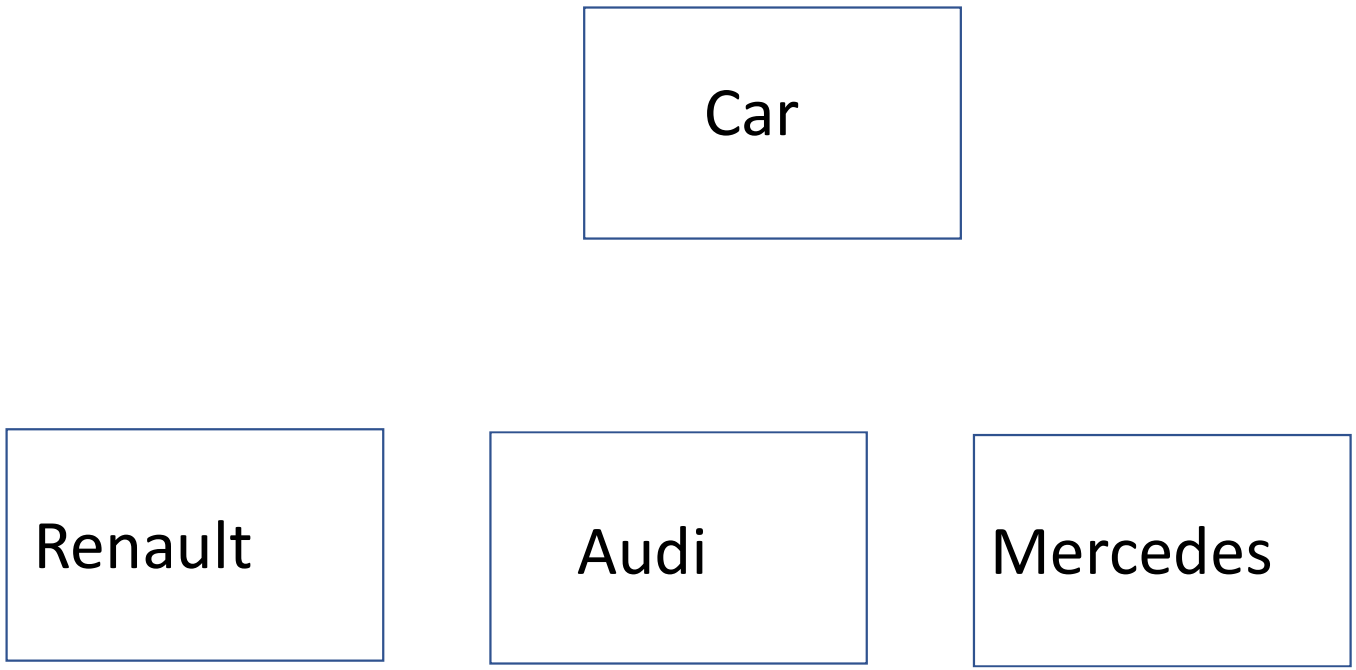
Java Variables

Following are the types of variables in Java:

- Local Variables
- Class Variables (Static Variables)
- Instance Variables (Non-static Variables)

Objects and classes

- A class is a definition about objects
- The attributes and methods in a class are thus declarations that do not contain values.
- objects are created instances of a class.
- Each has its own attributes and methods
- The values of the set of attributes describe the state of the objects.


Car
Renault
Audi
Mercedes

Objects in Java

- If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behaviour.
- If we consider a dog, then its state is - name, breed, colour, and the behaviour is - barking, wagging the tail, running.
- If you compare the software object with a real-world object, they have very similar characteristics.
- Software objects also have a state and a behaviour. A software object's state is stored in fields and behaviour is shown via methods.
- So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

Classes in Java

- A class is a blueprint from which individual objects are created.
- Following is a sample of a class.

```
public class Dog {
    String breed;
    int age;
    String color;

    void barking() {
    }

    void hungry() {
    }

    void sleeping() {
    }
}
```

- A class can contain any of the following variable types.
- **Local variables:** Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables:** Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables:** Class variables are variables declared within a class, outside any method, with the static keyword.
- A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

Constructors

- When discussing about classes, one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.
- Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.
- Syntax

```
class ClassName {
ClassName() { }
}
```

Creating an Object

There are three steps when creating an object from a class:

- **Declaration**: A variable declaration with a variable name with an object type.
- **Instantiation**: The 'new' keyword is used to create the object.
- **Initialization**: The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

- [Object type: [ClassName] variable= new [Constructor]();

- Example
- public class Book{
- public Book (){
- {
- }

- What about creating a new object of the Book Class?

- Book book= new Book();

```java
public class Book {
    public Book(){
        // this is constuctur has no parameters
        System.out.println("the title is empty");
    }
    public Book (String title){
        // this is constuctur has one parameter, "title"
        System.out.println("the title is:" + title);
    }
    public static void main(String [] args){
        Book book = new Book("Java");
        Book book1=new Book();


    }
}
```

Arrays

```
dataType[] arrayRefVar; // preferred way.
or
dataType arrayRefVar[]; // works but not preferred way.
```

```
double[] myList; // preferred way.
or
double myList[]; // works but not preferred way.
```

ArrayList

```
import java.util.ArrayList;
public class Main {
    public static void main(String[] args) {
        ArrayList<String> cars = new ArrayList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        System.out.println(cars);
    }
}
```

Loops

while loop Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

for loop Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

do...while loop Like a while statement, except that it tests the condition at the end of the loop body.

Control Statement

break statement Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.

continue statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Syntax

Following is the syntax of enhanced for loop:

```
for(declaration : expression) {
 // Statements }
```

```java
public class Test {
   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         System.out.print( x );
         System.out.print(",");
      }
      System.out.print("\n");
      String [] names = {"James", "Larry", "Tom", "Lacy"};

      for( String name : names ) {
         System.out.print( name );
         System.out.print(",");
      }
   }
}
```

```java
public class Calculattion {
    public static void main(String[] args) {
//  TODO Auto-generated method stub
        int sum = 0;
        for(int j = 1; j<=10; j++) {
            sum = sum + j;

        }
        System.out.println("The sum of first 10 natural numbers is " + sum)
    }
}
```

Syntax

```
while (condition) {
// code block to be executed
 }
```

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
```

Decision making

if statement An **if statement** consists of a boolean expression followed by one or more statements.

if...else statement An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is false.

nested if statement You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

switch statement A **switch** statement allows a variable to be tested for equality against a list of values.

Syntax

```
if(Boolean_expression) {
// Statements will execute if the Boolean expression is true
}
```

```
public class Test {
   public static void main(String args[]) {
      int x = 10;
      if( x < 20 ) {
         System.out.print("This is if statement");
      }
   }

}
```

```java
for (int i = 0; i < 10; i++) {
    if (i == 4) {
        break;
    }
    System.out.println(i);
}
```

The import statement

Syntax:

package <top_pkg_name>[.<sub_pkg_name>]*;

Usage:

- **import <pkg_name>[.<sub_pkg_name>]*.*;**

Examples:

import java.util.List;

import java.io.*;

Exceptions

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

        String [] names = {"James", "Larry", "Tom", "Lacy"};

                System.out.println(names[5]);
    }
}
```

Try to compile and execute the above program.

- An exception can occur for many different reasons. Following are some scenarios where an exception occurs:

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

- Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Catching Exceptions

- A method catches an exception using a combination of the **try** and **catch** keywords.
- A try/catch block is placed around the code that might generate an exception.
- Code within a try/catch block is referred to as *protected code*, and the syntax for using try/catch looks like the following:

```
try {
// Protected code
} catch (ExceptionName e1)
{
// Catch block
}
```

Example

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

    try {
        String[] names = {"James", "Larry", "Tom", "Lacy"};

        System.out.println(names[5]);
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Exception Thrown :" + e);}
    System.out.println("out of the block");
    }
    }
```

Output

```java
import java.io.File;
import java.io.FileReader;


public class FilenotFound_Demo {
    public static void main(String args[]) {
        File file = new File("E://file.txt");
        FileReader fr = new FileReader(file);
    }
}
```

Multiple Catch Blocks

- A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following:
- Syntax

```
try {
// Protected code } catch (ExceptionType1 e1) {
// Catch block } catch (ExceptionType2 e2) {
 // Catch block
} catch (ExceptionType3 e3) {
// Catch block }
```

Example

```
try {

    File file = new File("E://file.text");
    FileReader fr = new FileReader(file);
} catch (FileNotFoundException e){
    System.out.println("Exception thrown" + e);
}
```

Thank you