# Java Basics

## 1 Overview

This lab is an introduction to Java programming language. It is designed for the students who have no experience in Java. This lab is more about understanding the code than writing your own. However, students are encouraged to install relevant software packages (JDK & Eclipse) and write/edit/compile their code.

## 2 Lab Environment (Windows)

**Installing JDK** http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
Download Java SE Development Kit, run the installer and follow the directions. Accept the default settings.

**Installing Eclipse** http://www.eclipse.org/downloads/
Run the installer and follow the directions. Accept the default settings.

## 3 Introduction to Java

### 3.1 Types, variables and operators

A Java program structure is as follows:

```
class CLASSNAME{
 public static void main(String[] arguments){
  STATEMENTS
  }
}
```

For example, a simple program:

```
class Hello{
 public static void main(String[] arguments){
  System.out.println("Hello World."); //Print once
  System.out.println("Line no.2");    //Print again
  }
}
```

- **Types** Kinds of values that can be stored and manipulated
  *boolean*: Truth value (true or false).
  *int*: Integer (0, 1, -47).
  *double*: Real number (3.14, 1.0, -2.1).
  *String*: Text ("hello", "example").

- **Variables**
  Form: TYPE NAME
  Example: String foo = "IAP 6.092";

- **Operators**
  Symbols that perform simple computations:
  Assignment: =
  Addition: +
  Subtraction: -
  Multiplication: *
  Division: /

  Order of operation is as follows:
  1. Parentheses
  2. Multiplication and division
  3. Addition and subtraction

  Examples:

```
class DoMath{
 public static void main(String[] arguments){
  double score = 1.0 + 2.0 * 3.0;
  System.out.println(score);
  double copy = score;
  copy = copy / 2.0;
  System.out.println(copy);
  System.out.println(score);
  }
}

What would be printed?
```

```
//the following program is about string concatenation
class StringConcate{
 public static void main(String[] arguments){
  String text = "hello" + " world";
  text = text + " number " + 5;  //text = "hello world number 5"
  System.out.println(text);
  }
```

```
}
```

## 3.2 Methods and Conditionals

- **Methods**

  A method is a set of code which is referred to by name and can be called (invoked) at any point in a program simply by utilizing the method's name. Think of a method as a subprogram that acts on data and often returns a value.
  Example:

```
class NewLine{
 public static void newLine(){
  System.out.println("");
  }

 public static void threeLines() {
  newLine(); newLine(); newLine();
  }

  public static void main(String[] arguments){
   System.out.println("Line 1");
   threeLines();
   System.out.println("Line 2");
  }
}
```

  **Parameters**: In Java, parameters sent to methods are passed-by-value. Clarification: What is passed "to" a method is referred to as an "argument". The "type" of data that a method can receive is referred to as a "parameter".

  Example:

```
class Square{
 public static void printSquare(int x){
  System.out.println(x*x);
  }

  public static void main(String[] arguments){
   int value = 2;
       printSquare(value);
       printSquare(3);
       printSquare(value*2);
  }
}
```

- **Conditionals** Java is equipped with a selection operator that allows us to construct a conditional expression. The use of a conditional expression can in some cases simplify the code with respect to the use of an if-else statement. Condition is a boolean expression. Example:

```
class conditionals{
 public static void test(int x){
  if (x > 5){
  System.out.println(x + " is > 5");
  } else if (x == 5){
  System.out.println(x + " equals 5");
  } else {
  System.out.println(x + " is < 5");
  }
 }

  public static void main(String[] arguments){
   test(6);
       test(5);
       test(4);
  }
}
```

## 3.3 Loops and Arrays

- **Loops**

  Loop operators allow to loop through a block of code. Following are examples of *for* and *while* loop.

  *while* loop:

```
int i = 0;
while (i < 3) {
System.out.println("Rule #" + i);
i = i+1;
}
```

  *for* loop:

```
for (int i = 0; i < 3; i=i+1) {
System.out.println("Rule #" + i);
}
```

  **Branching**: ***break*** terminates a ***for*** or ***while*** loop, and ***continue*** skips the current iteration of a loop and proceeds directly to the next iteration.

```
for (int i=0; i<100; i++) {
 if(i == 50)
 break;
System.out.println("Rule #" + i);
}
-----------------------------------------
fo (int i=0; i<100; i++) {
 if(i == 50)
 continue;
System.out.println("Rule #" + i);
}
```

- **Arrays** An array is an indexed list of values. You can make an array of any type int, double, String, etc..All elements of an array must have the same type. To create an array of a given size, use the operator new:

```
int[] values = new int[5];
```

Curly braces can be used to initialize an array. It can ONLY be used when you declare the variable:

```
int[] values = { 12, 24, -23, 47 };
```

More Examples:

```
int[] values = new int[12];
int size = values.length; // 12
int[] values2 = 1,2,3,4,5
int size2 = values2.length; // 5
```

```
int[] values = new int[5];
int i = 0;
while (i < values.length) {
 values[i] = i;
 int y = values[i] * values[i];
 System.out.println(y);
 i++;
}
```

## 3.4 Objects and Classes

- **Objects**: Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging the tail, barking, eating. An object is an instance of a class.

- **Classes**: A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support. 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() {
   }
}
```

**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. Example:

```
public class Puppy {
   public Puppy() {
   }
   public Puppy(String name) {
      // This constructor has one parameter, name.
   }
}
```

**Creating an Object** As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects. There are three steps when creating an object from a class:

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

Following is an example of creating an object:

```
public class Puppy {
   public Puppy(String name) {
      // This constructor has one parameter, name.
      System.out.println("Passed Name is :" + name );
   }
```

```
   public static void main(String []args) {
      // Following statement would create an object myPuppy
      Puppy myPuppy = new Puppy( "tommy" );
   }
}

Output: Passed Name is: tommy
```

The following example explains how to access instance variables and methods of a class.

```
public class Puppy {
   int puppyAge;

   public Puppy(String name) {
      // This constructor has one parameter, name.
      System.out.println("Name chosen is :" + name );
   }

   public void setAge( int age ) {
      puppyAge = age;
   }

   public int getAge( ) {
      System.out.println("Puppy's age is :" + puppyAge );
      return puppyAge;
   }

   public static void main(String []args) {
      /* Object creation */
      Puppy myPuppy = new Puppy( "tommy" );

      /* Call class method to set puppy's age */
      myPuppy.setAge( 2 );

      /* Call another class method to get puppy's age */
      myPuppy.getAge( );

      /* You can access instance variable as follows as well */
      System.out.println("Variable Value :" + myPuppy.puppyAge );
   }
}

Output:
Name chosen is :tommy
```

```
Puppy's age is :2
Variable Value :2
```

## 3.5 Access Modifiers and Class Scope

- **Access Modifiers**

  The first (left-most) modifier used lets you control what other classes have access to a member field. For the moment, consider only public and private.

  **public modifier**: the field is accessible from all classes. **private modifier**: the field is accessible only within its own class. Have a look at the following code:

```
public class CreditCard{
  String cardNumber;
  double expenses;
  void charge(double amount){
   expenses = expenses + amount;
  }
  String getCardNumber(String password){
  if (password.equals("secret!3*!")){
  return cardNumber;
  }
  return "jerkface";
  }
}
```

  An adversary can write the following program:

```
public class Malicious {
public static void main(String[] args) {
maliciousMethod(new CreditCard());
}
static void maliciousMethod(CreditCard card)
 {
  card.expenses = 0;
  System.out.println(card.cardNumber);
 }
}
```

  The correct way of using access modifiers will be "private" for both *cardNumber* and *expenses*. We should hide variables or some implementation that may be changed so often in a class to prevent outsiders access it directly, they must access it via getter and setter methods, this is called **encapsulation**.

- **Class Scope:** Just like Methods, variables are accessible intside {}. In the following code, only Method level *serving* is updated.

```
public class Baby {
int servings;
void feed(int servings) {
 servings = servings + servings;
}
void poop() {
 System.out.println("All better!");
 servings = 0;
 }
}
```

**"this"** keyword: Clarifies scope, it means my object. Above code can be written as:

```
public class Baby {
int servings;
void feed(int servings) {
this.servings =
this.servings + servings;
}
void poop() {
 System.out.println("All better!");
 servings = 0;
 }
}
```

### 3.6 Inheritance, Exceptions and File I/O

- **Inheritance**: Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class. ... The subclass can add its own fields and methods in addition to the superclass fields and methods. For example:

```
public class Dude {
 public String name;
 public int hp = 100
 public int mp = 0;
 public void sayName() {
 System.out.println(name);
}
 public void punchFace(Dude target) {
 target.hp -= 10;
 }
}
```

Now create a wizard:

```
public class Wizard {
// ugh, gotta copy and paste
// Dude's stuff
}
```

Now augment a wizard:

```
public class Wizard extends Dude {
 ArrayList<Spell> spells;
 public class cast(String spell) {
 // cool stuff here
 ...

 mp -= 10;
 }
}
```

What about grand Wizard?

```
public class GrandWizard extends Wizard {
 public void sayName() {
 System.out.println("Grand wizard" + name)
 }
}
grandWizard1.name = "Flash"
grandWizard1.sayName();
((Dude)grandWizard1).sayName();
```

What Java does when it sees
*grandWizard1.punchFace(dude1)*

1. Look for punchFace()in the GrandWizard class
2. It's not there! Does GrandWizard have a parent?
3. Look for punchFace()in Wizard class
4. It's not there! Does Wizard have a parent?
5. Look for punchFace()in Dude class
6. Found it! Call punchFace()
7. Deduct hp from dude1

- **Exceptions**: We use exceptions when we need to tell the code using your method that something went wrong. You can catch an exception:

```
try {
get(-1);
```

```
} catch (ArrayOutOfBoundsException err) {
System.out.println("oh dear!");
}
```

or Rethrow it:

```
void doBad() throws ArrayOutOfBoundsException {
get(-1);
}
```

If no one catches an exception, java will print an error message.

- **File I/O** The following code reads a file from the system:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class ReadFile {
 public static void main(String[] args) throws IOException {
  // Path names are relative to project directory (Eclipse Quirk )
  FileReader fr = new FileReader("./src/readme");
  BufferedReader br = new BufferedReader(fr);
  String line = null;
  while ((line = br.readLine()) != null) {
  System.out.println(line);
  }
  br.close();
 }
}
```