CORE JAVA PROGRAMMING BASICS

By www.HadoopExam.com

Note: These instructions should be used with the HadoopExam Apache OOzie: Professional Trainings. Where it is executed and you can do hands on with trainer.

- **JDK: Java Development Kit**
- **JRE : Java Runtime Environment**
- **Creating WorkSpace in eclipse.**
- **Creating Projects in eclipse**
- **Class**: It is a building block of Java Programming

In class you describe all the parts and characteristics of one of those building blocks.

Sample Welcome Class

```
public class Welcome {

	public static void main(String[] args) {
		System.out.println("Welcome to HadoopExam Learning Resources : Core Java Training");
	}
}
```

Object: Most of the time, you will be creating objects from the class. An *object* is a runtime instance of a class in memory.

Fields and Methods: You will be having these two components in a Java class

- **Methods** also known as functions e.g. totalProducts , operate on that state
- **Fields** also known as variables. e.g. **webSiteName ,** it shows the state of the program.
- **Keywords** : public, class

```
public class Company {

	String webSiteName="www.HadoopExam.com";

	public int totalProducts() {
		System.out.println("We have 30 Products in total");
		return 30;
	}
}
```

- Discuss return type.
- Shows behaviors of public statement, using below class

```
public class Welcome {

	public static void main(String[] args) {
		System.out.println("Welcome to HadoopExam Learning Resources : Core Java Training");

		Company myCompany = new Company();

		System.out.println(myCompany.webSiteName + " has total " + myCompany.totalProducts() + " Products
");

	}
}
```

- Create getter and setter methods in eclipse and use it in Welcome class.

```
public class Company {

	String webSiteName="www.HadoopExam.com";

	public String getWebSiteName() {
		return webSiteName;
	}

	public void setWebSiteName(String webSiteName) {
		this.webSiteName = webSiteName;
	}

	public int totalProducts() {
		return 30;
	}

	public void printInfo(){
		System.out.println("We have 30 Products in total");
	}
}
```

Method Signature:

```
public void setWebSiteName(String webSiteName)
```

Discuss about comments: (Watch Video)

Relation between Class and File: If you keen below code in a single file named Welcome.java then it is Ok.

Some Rule

- At most one of the classes in the file is allowed to be public
- If you do have a public class, it needs to match the file name

```
public class Welcome {

	public static void main(String[] args) {
		System.out.println("Welcome to HadoopExam Learning Resources : Core Java Training");
		Company myCompany = new Company();
		System.out.println(myCompany.webSiteName + " has total " + myCompany.totalProducts() + " Products ");

	}
}

class InnerCompany {

	String webSiteName="www.HadoopExam.com";

	public String getWebSiteName() {
		return webSiteName;
	}

}
```

About main() method : Entry Point for a Java Process

- A Java program begins execution with its main() *method*
- A main() method is the gateway between the startup of a Java process, which is managed by the *Java Virtual Machine* (JVM), and the beginning of the programmer's code.

Run Code using Command Line

```
javac Welcome.java (Which will generate bytecode in file named Welcome.class)
java Welcome
```

- Bytecode consists of instructions that the JVM knows how to execute.

Supply Arguments to Class and use it.

```
public class Welcome {

	public static void main(String[] args) {
		System.out.println("Welcome " + args[0] );
		System.out.println("Welcome " + args[1] );
		System.out.println("Welcome " + args[2] );
	}
}
```

- Java comes with thousands of built-in classes
- Packages: Java puts classes in *packages*. These are logical groupings for classes
- Visit Random class source code in eclipse (Watch Video)
- Import statements tell Java which packages to look in for classes

```
//import java.util.Random;

public class Welcome {

	public static void main(String[] args) {
		System.out.println(Integer.MAX_VALUE); //Imported by default
		Random randomValue = new Random(); //Require explicit import of package
```


```
            System.out.println(randomValue.nextInt(100));
        }
}
```

- Put Company class in a package and use it in Welcome class.

```
import com.hadoopexam.Company;
public class Welcome {
   public static void main(String[] args) {
               System.out.println("Welcome to HadoopExam Learning Resources : Core Java Training");
               Company myCompany = new Company();
               System.out.println(myCompany.webSiteName + " has total " + myCompany.totalProducts() + " Products ");
   }
}
```

- Wildcards: Classes in the same package are often imported together.

```
import com.hadoopexam.*;
```

- Default package , which will be always imported

```
Import java.lang.*;
```

- Redundant Imports : (Watch Video)
- **Wrong imports**

import java.*;	NO GOOD – a wildcard only matches class names, not "lang.*Files"
import java.*.*;	NO GOOD – you can only have one wildcard and it must be at the end
import com.hadoopexam.Company.*;	NO GOOD – you cannot import methods only class names

- Because of the package, class name should not be unique across all packages.
- There are two classes with same name in different packages.

java.sql.Date java.util.Date

- Following code will give error.

```
import com.hadoopexam.Company;
import java.sql.Date;
import java.util.Date;
```


```
public class Welcome {
	public static void main(String[] args) {
			System.out.println(new Date());
	}
}
```

Valid Example

```
import com.hadoopexam.Company;
import java.sql.Date;

public class Welcome {
	public static void main(String[] args) {
			System.out.println(new Date(10000000));
			System.out.println(new java.util.Date(10000000));
	}
}
```

Constructor: It is a special type of Method.

- Creating an instance of a Class

```
Company myCompany = new Comapny();
```

- new Company is creating actual instance of a class
- myCompany is a reference to newly created object.

```
package com.hadoopexam;
public class Company {

	public String webSiteName="www.HadoopExam.com";

	Company(){}		//Default Constructor

	public int totalProducts() {
		return 30;
	}

	public void printInfo(){
```


```
                System.out.println("We have 30 Products in total");
        }
    }
```

- Invalid definition of constructor

```
public void Company(){} //it is a another method.
```

- Do nothing constructor is defined by a compiler itself.
- The purpose of a constructor is to initialize fields, although you can put any code in there.

```
import com.hadoopexam.Company;

public class Welcome {
      public static void main(String[] args) {
      System.out.println("Welcome to HadoopExam Learning Resources : Core Java Training");
      Company myCompany = new Company();
      System.out.println(myCompany.webSiteName + " has total " + myCompany.totalProducts() + " Products ");

      Company anotherCompany = new Company("QuickTechie.com" , 19);
      System.out.println(anotherCompany.webSiteName + " has " + anotherCompany.totalProducts() + " Products ");

      }
}
```

```
package com.hadoopexam;
public class Company {
      public String webSiteName;
      public int totalProducts;

      public Company(){
            webSiteName="www.HadoopExam.com";
            totalProducts=30;
      }

      public Company(String name, int productCount){
            webSiteName=name;
            totalProducts=productCount;
      }

      public int totalProducts() {
```


```
            return totalProducts;
      }

      public void printInfo(){
            System.out.println("We have "+ totalProducts +" Products in total");
      }
}
```

Basic Rule of initialization.

- Fields and instance initializer blocks are run in the order in which they appear in the file.
- The constructor runs after all fields and instance initializer blocks have run.
- Order matters for the fields and blocks of code. You can't refer to a variable before it has been initialized
-

```
public class Welcome {
      private String name = "HadoopExam.com";
      {
            System.out.println("It is the block 1 " + name);
      }
      {
            System.out.println("It is the block 2 " + (name = "Training4Exam.com"));
      }
      Welcome() {
            name = "QuickTechie.com";
      }
      public static void main(String[] args) {
            Welcome welcome = new Welcome();
            System.out.println(welcome.name);
      }
}
```

Module 5: Java applications contain two types of data: primitive types and reference types.

Primitive Types: Java has eight built-in data types, referred to as the Java *primitive types*.

Keyword	Type		Example
boolean	TRUE or FALSE		TRUE
byte	8-bit	integer	123

short	16-bit	integer	123
int	32-bit	integer	123
long	64-bit	integer	123
float	32-bit	floating-point	123.45f
double	64-bit	floating-point	123.456
char	16-bit	Unicode	'a'

- A float requires the letter f following the number so Java knows it is a float.
- byte, short, int, and long are used for numbers without decimal points.
- Java allocates 32 bits if you write this: **int value**;
- long max = 3333333333; // DOES NOT COMPILE
- long max = 3333333333L; // DOES COMPILE
- When a number is present in the code, it is called a *literal*
- By default, Java assumes you are defining an int value with a literal
- primitive types that hold their values in the memory where the variable is allocated

One more thing you need to know about numeric literals is a feature added in Java 7. You can have underscores in numbers to make them easier to read:

```
double notAtStart = _1000.00; // DOES NOT COMPILE
double notAtEnd = 1000.00_; // DOES NOT COMPILE
double notByDecimal = 1000_.00; // DOES NOT COMPILE
double annoyingButLegal = 1_00_0.0_0; // this one compiles
```

Reference Types:

- A *reference type* refers to an object (an instance of a class).
- References do not hold the value of the object they refer to.
- Instead, a reference "points" to an object by storing the memory address where the object is located, a concept referred to as a *pointer*.
- You can only use the reference to refer to the object.

Some important points
- A reference can be assigned to another object of the same type.
- A reference can be assigned to a new object using the new keyword.

```
public class Welcome {
	String name;
	Welcome(String siteName) {
		name = siteName;
```


```
        }
        public static void main(String[] args) {
                Welcome welcome = new Welcome("HadoopExam.com");
                welcome = new Welcome("QuickTechie.com");
                System.out.println(welcome.name);
        }
}
```

Please remember:

- Reference types can be assigned null, which means they do not currently refer to an object.
  int value = null; // DOES NOT COMPILE
  String s = null;
- Reference types can be used to call methods when they do not point to null.
- Primitives do not have methods declared on them.
- All the primitive types have lowercase type names.
- All classes that come with Java begin with uppercase.

About Variables:

- A *variable* is a name for a piece of memory that stores data. When you declare a variable, you need to state the variable type along with giving it a name.

```
public String webSiteName;
public int totalProducts;
```

- **Initializing variables**

```
    webSiteName="www.HadoopExam.com";
    totalProducts=30;
```

- **Another way of declaring variables**

```
    String s1, s2;
    String s3 = "Hadoop", s4 = "Exam";
    String i1, i2, i3 = "HadoopExam.com";
    int num, String name; // DOES NOT COMPILE
```

About identifiers: There are only three rules to remember for legal identifiers

- The name must begin with a letter or the symbol $ or _.
- Subsequent characters may also be numbers.

- You cannot use the same name as a Java *reserved word*.
- Some example of illegal identifiers
  - 1Exam // identifiers cannot begin with a number
  - hadoop@exam // @ is not a letter, digit, $ or _
  - *$hadoop // * is not a letter, digit, $ or _
  - public // public is a reserved word

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	FALSE	final	finally
float	for	goto	if	implements
import	instanceof	int	interface	long
native	new	null	package	private
protected	public	return	short	static
strictfp	super	switch	synchronized	this
throw	throws	transient	TRUE	try
void	volatile	while		

Variables Initialization: Before you can use a variable, it needs a value.

Local Variables:

- A *local variable* is a variable defined within a method.
- Local variables must be initialized before use.
- The compiler will not let you read an uninitialized value.
- Below code will not compile.

```
public void doCalculation(){
        int i;
        int j;
```


```
            System.out.println( "Final Calculation": i + j);
      }
```

- The compiler is smart enough to recognize variables that have been initialized after their declaration but before they are used.

Instance and Class Variables:

- Variables that are not local variables are known as *instance variables* or *class variables.*
- Instance variables are also called fields.
- Class variables are shared across multiple objects.
- You can tell a variable is a class variable because it has the keyword static before it.
- Instance and class variables do not require you to initialize them.
- As soon as you declare these variables, they are given a default value.

```
public class Welcome {
   String name;
   char ch;
   boolean b;
   byte by;
   short sh;
   int i;
   long ln;
   float fl;
   double db;
   {
         System.out.println("String value " + name);
         System.out.println("Char value " + ch); //'\u0000' (NUL)
         System.out.println("boolean value " + b);
         System.out.println("byte value " + by);
         System.out.println("short value " + sh);
         System.out.println("int value " + i);
         System.out.println("long value " + ln);
         System.out.println("float value " + fl);
         System.out.println("double value " + db);
   }

   Welcome(String siteName) {
         name = siteName;
   }

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


```
            Welcome welcome = new Welcome("HadoopExam.com");
      }
   }
```

Variable Scope:

```
public void greetMe(String name){ //method parameter
            String str= "Welcome";
      }
```

- Both of these variables are said to have a *scope* local to the method.
- This means they cannot be used outside the method.
- Local variables can never have a scope larger than the method they are defined in. However, they can have a smaller scope.

```
      public void greetMe(String name){
            if(name!=null){
                  String str= "Welcome";
            }
            System.out.println(str + name); //str not available
      }
```

- More about different types of variable

```
   public class Welcome {
      static int HEIGHT = 130;  //Class Variable
      String name; //instance variable

      //local variable : name, myHeight, str,profstr
      public void greetMe(String name, int myHeight){
            if(myHeight < HEIGHT){
                  String str= "Welcome Student " + name;
                  System.out.println(str);
            }
            else{
                  String profstr= "Welcome Professor " + name;
                  System.out.println(profstr);
            }
      }
   }
```

- Local variables- in scope from declaration to end of block
- Instance variables - in scope from declaration until object garbage collected
- Class variables - in scope from declaration until program ends

Various statements order in a Java Class

Element	Example	Required?	Where does it go?
Package declaration	package com.hadoopexam;	No	First line in the file
Import statements	import java.util.*;	No	Immediately after the package (Empty lines allowed before that)
Class declaration	public class Welcome	Yes	Immediately after the import (Empty lines allowed before that)
Field declarations	String name="HadoopExam";	No	Anywhere inside a class
Method declarations	public void doCalculate()	No	Anywhere inside a class

- A file is also allowed to have neither class be public. As long as there isn't more than one public class in a file, it is okay.

Garbage Collection: Destroying Object (Cleaning memory area)

- All java objects are stored in Java heap.
- The heap may be quite large, depending on your environment, but there is always a limit to its size. If your program keeps instantiating objects and leaving them on the heap, eventually it will run out of memory.
- Garbage collection refers to the process of automatically freeing memory on the heap by deleting objects that are no longer reachable in your program.
- Sytem.gc() : (Just a suggestion to JVM) it is not guaranteed to run, and you should be able to recognize when objects become eligible for garbage collection.
- An object is no longer reachable when one of two situations occurs:
    - The object no longer has any references pointing to it.
    - All references to the object have gone out of scope

- **Reference**: A reference may or may not be created on the heap. All references are the same size, no matter what their data type is, and are accessed by their variable name.
- **Object**: Objects are always on the heap. They have no name and can only be accessed via a reference. Objects vary in size depending on their class definition.
- Draw object reference diagram for below code.


```
public static void main(String[] args) {
        String first, second;
        first = new String("HadoopExam");
        second = new String("QuickTechie");
        first = second;
        String three = first;
        first = null;
}
```

finalize() method :

- finalize() is only run when the object is eligible for garbage collection.
- Java allows objects to implement a method called finalize() that might get called.
- This method gets called if the garbage collector tries to collect the object.
- If the garbage collector doesn't run, the method doesn't get called.
- If the garbage collector fails to collect the object and tries to run it again later, the method doesn't get called a second time.
- you are highly unlikely to use it in real projects.
- finalize() call could run zero or one time.

Java is Secure :

- Java code runs inside the JVM. This creates a sandbox that makes it hard for Java code to do evil things to the computer it is running on.
-

All Products List of www.HadoopExam.com

TRAINING'S

- Hadoop BigData Professional Training (3500INR/$79)
- HBase (NoSQL) Professional Training (3500INR/$79)
- Apache Spark Professional Training (~~3900INR/$89~~ for a week 3500INR/$75)
- Apache OOZie (Hadoop workflow) Professional Training
- Beginner AWS Training Course- **(HETRNAWS101)**

MAPR HADOOP AND NOSQL CERTIFICATION

- **MapR Hadoop Developer Certification**
- **MapR HBase NoSQL Certification**
- **MapR Spark Developer Certification (In Progress)**

CLOUDERA HADOOP AND NOSQL CERTIFICATION

- CCA50X : Hadoop Administrator
- CCA-175 Cloudera® (Hadoop and Spark Developer)
- CCP:DE575 : Cloudera® Data Engineer Certification

DATABRICKSA OREILLY SPARK CERTIFICATION

- Apache Spark Developer

AWS: AMAZON WEBSERVICE CERTIFICATION

- AWS Solution Architect : Associate
- AWS Solution Architect: Professional
- AWS Developer : Associate
- AWS Sysops Admin : Associate

MICROSOFT AZURE CERTIFICATION

- Azure 70-532
- Azure 70-533

DATA SCIENCE CERTIFICATION

- EMC E20-007

EMC CERTIFICATIONS

- EMC E20-007

SAS ANALYTICS CERTIFICATION

- SAS Base A00-211
- SAS Advanced A00-212
- SAS Analytics : A00-240
- SAS Administrator : A00-250

ORACLE JAVA CERTIFICATION

- Java 1z0-808
- Java 1z0-809

ORACLE DATABASE CLOUD CERTIFICATION

- 1z0-060 (Oracle 12c)
- 1z0-061 (Oracle 12c)