Java & Linux

Building blocks of Hadoop

Apache Hadoop

- Open source platform built on two technologies
  - Written in Java programming language.
  - Runs on Linux operating system

Java

Storing, analysing and processing large data sets.

Installing and Configuring Java and Eclipse

- Java
  - Available for Windows / Linux / MacOS etc
  - Download from http://www.oracle.com/
- Eclipse
  - Integrated Development Environment (IDE) which is used for building applications in languages like Java, C, C++, C#, etc
  - Available for Windows / Linux / MacOS etc
  - Download from http://www.eclipse.org/downloads/eclipse-packages/

How much Java to learn?

Since Hadoop is written in Java, this knowledge of Java basics if essential to learn Hadoop

- Class & Object
- Variables & Data Types
- Arrays
- Inheritance & Interfaces
- Control Flows
- Exception Handling
- Serialization & Deserialization

Class & Object

Class

Class is a blueprint that defines the behavior and property of an object i.e. it is the template for an object. Every class has a **state** and **behavior**.

Example: Bicycles have **State**

- current gear
- current speed
- current pedal, etc

Bicycles have **Behaviour**

- change pedal
- change gear
- apply brakes, etc

Bicycle Class

```
public class Bicycle {

    final String manufacturerName; // Single line comment
    final float bicycleHeight, bicycleWeight;
    final int[] gears;

    /*
       Multi line comment
       Constructor
     */
    public Bicycle(String manufacturerName, float bicycleHeight, float bicycleWeight, int[] gears) {
       this.manufacturerName = manufacturerName;
       this.bicycleHeight = bicycleHeight;
       this.bicycleWeight = bicycleWeight;
       this.gears = gears;
    }

    int currentGear;
    float currentSpeed, currentDistance; // Single line declaration

    public void accelerate(){  currentSpeed ++; }

    public void brake(){ currentSpeed --;  }

    public void changeGear(){ currentGear = gears[(int )(Math. random() * gears.length + 1)]; }

}
```

Object

It is an instance or copy of the blueprint known as Class. For programmers to use the **state** and the **behavior** of a particular class, it is necessary to create an object. An object of a class has all the attributes bundled in it. Identifying the **state** and **behavior** for an object is crucial when designing a class for a real world application.

Bicycle Object

```
public class BicycleObject {
    public static void main(String[] args) {
        int[] gears = {1,2,3,4,5,6};
        Bicycle bicycle = new Bicycle("BSA", 27*2.54f, 20.0f, gears);

        bicycle.accelerate();

    }
}
```

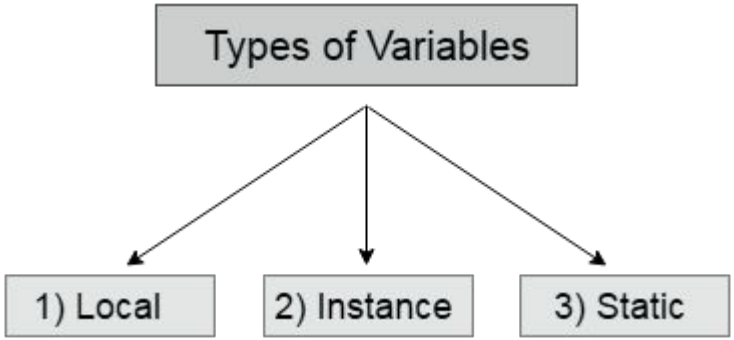
Execution

main() method in Java is static because it can then be invoked by the runtime engine without having to instantiate an instance of the parent class.

Variables & Data types

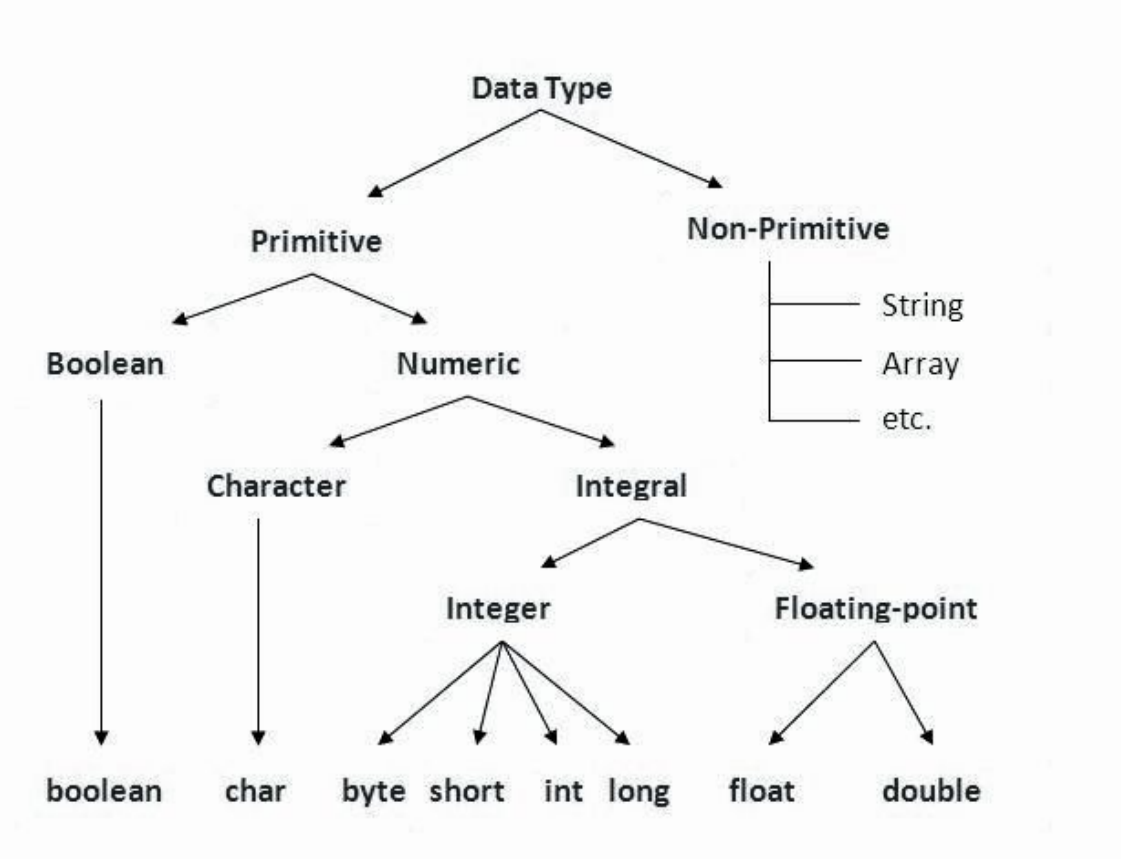
Variables

Variable is a name of memory location.




```
public class BSABicycle {

    String manufacturerName = "BSA"; // instance variable
    static String cycleModel = "mountainbike"; // static variable

    public void accelerate(){
          int step = 1; // local variable
          currentSpeed = currentSpeed + step;
    }

}
```

DataTypes


Data Type
Primitive
Non-Primitive
String
Array
etc.
Boolean
Numeric
Character
Integral
Integer
Floating-point
boolean
char
byte
short
int
long
float
double

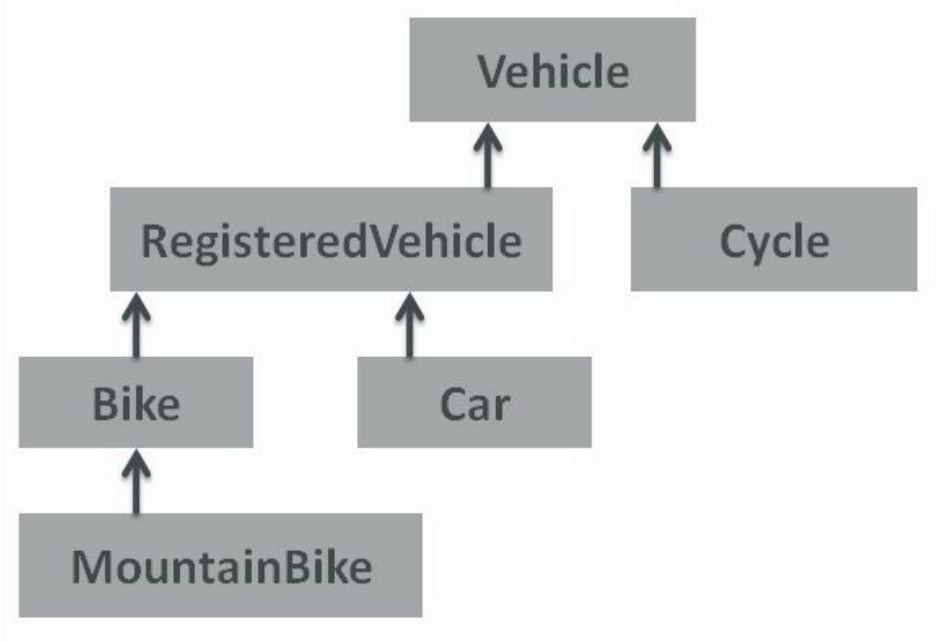
Examples

- Sum
- Widening
- Narrowing

Inheritance & Interface

Inheritance

The process of **sharing the properties and behavior of one class to another** is known as Inheritance. A class that inherits (derives) the properties and behavior of another class is referred to as the Child Class or Subclass or derived class. The class that extends its properties and behavior to the child class is known as the Parent class or Superclass.



IS - A relationship

```
public class Animal{
}

public class Mammal extends Animal{
}

public class Reptile extends Animal{
}

public class Dog extends Mammal{
}
```

Now, based on the above example, In Object Oriented terms, the following are true:

- Animal is the superclass of Mammal class.
- Animal is the superclass of Reptile class.
- Mammal and Reptile are subclasses of Animal class.
- Dog is the subclass of both Mammal and Animal classes.

Now, if we consider the IS-A relationship, we can say:

- Mammal IS-A Animal
- Reptile IS-A Animal
- Dog IS-A Mammal
- Hence : Dog IS-A Animal as well

Abstract Class

A class that is **declared with abstract keyword**, is known as abstract class in java.

It can have abstract and non-abstract methods (method with body).

```
public abstract class BicycleFramework {
    public abstract void accelerate();
}
```

```
public class Bicycle extends BicycleFramework{
    // ------
    @Override
    public void accelerate(){
        currentSpeed ++;
    }

}
```

What is Abstraction?

Abstraction is a process of **hiding the implementation details and showing only functionality to the user**. It shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message.

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

- Abstract class (0 to 100%)
- Interface (100%)

Method Overloading

```
public class Main{
        public void sum(int a, int b){
                System.out.println(a+b);
        }
        public void sum(double a, double b){
                System.out.println(a+b);
        }

        public static void main(String args[]){
                Main test = new Main();
                test.sum(10,20);
                test.sum(10.5,20.5);
        }
}
```

Method Overloading (Compile time polymorphism)

- If a class have multiple methods by same name but different parameter is known as Method Overloading.
- appear in the same class or a subclass have the same name but, have different parameter lists, and, can have different return types

Advantage of Method Overloading

- Method overloading increases the readability of program.

There are two way to overload the method.

- By changing number of arguments.
- By changing the data type of arguments.

Method Overriding

```
class Vehicle{
        void run(){System.out.println("Vehicle is running");}
}
class Bike extends Vehicle{
        void run(){System.out.println("Bike is running safely");}
public static void main(String args[]){
        Bike obj = new Bike();
        obj.run();
}
```

Method Overloading (Compile time polymorphism)

- If a class have multiple methods by same name but different parameter is known as Method Overloading.
- appear in the same class or a subclass have the same name but, have different parameter lists, and, can have different return types

Advantage of Method Overloading

- Method overloading increases the readability of program.

There are two way to overload the method.

- By changing number of arguments.
- By changing the data type of arguments.

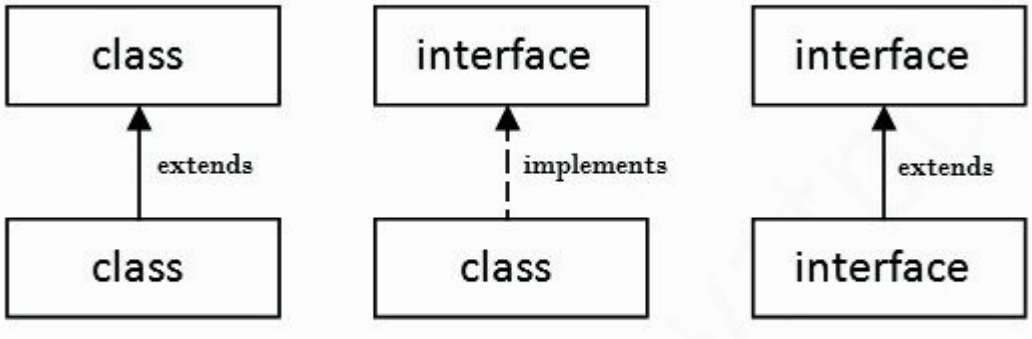
Interface

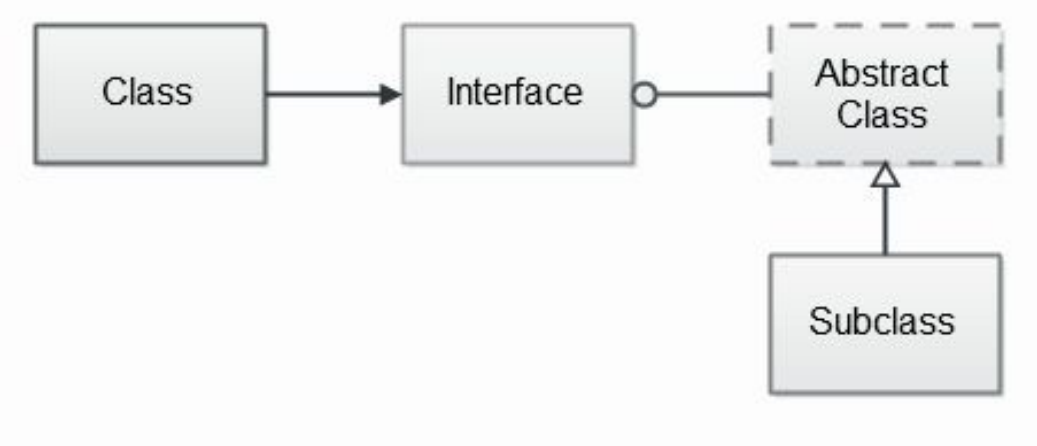
An **Interface in java** is a blueprint of a class. It has static constants and abstract methods.

The interface in java is a **mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.

Java Interface also represents **IS-A relationship**.

It cannot be instantiated just like abstract class.


class
extends
class
interface
implements
class
interface
extends
interface


Class
Interface
Abstract Class
Subclass

ClassA

```
public class ClassA {
	public String getMessage() {
		return "Hello";
	}
}
```

InterfaceA

```
public interface InterfaceA {
	default String getMessage() {
		return "Hola";
	}
}
```

DemoClass


```
public class DemoClass extends ClassA implements InterfaceA {

	public static void main(String[] args) {
		System.out.println(new DemoClass().getMessage());
	}

}
```

Significance of Interfaces in Java

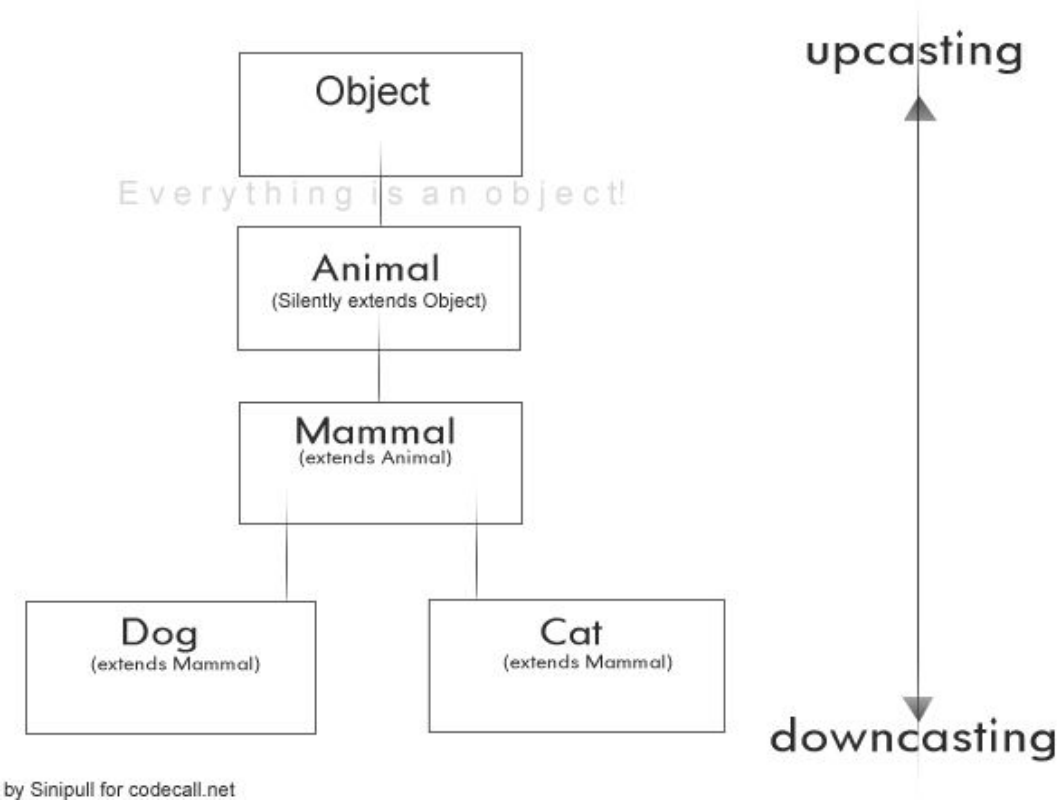
- **Interface acts as an agreement** for all child classes that want to inherit it.
- **Interfaces are a great starting point to define Type and create top level hierarchy**. At later point of time, if something else is needed it can be added to child classes.
- **Multiple inheritance**: Extending multiple super entities is called Multiple Inheritance. In Java a class is not allowed to extend multiple classes. To achieve multiple inheritance in Java, interfaces can be used means that a class can access various variables declared in more than one interface. Also it can provide implementation for the methods which can be inherited from multiple interfaces.
  “Implements” is the keyword used to inherit from an interface
- **Polymorphism**: Having different flavors of the same method is referred to as Polymorphism. This simply means that the method with same name can behave differently at different places. With interfaces, different classes can provide separate implementation of the same method declared in the interface.

Characteristics of an Interface

- An interface can **only hold a method's declaration but not its complete implementation**. This means that you can define the signature and template for a method but can't fill it with any logic. Such methods are known as Abstract methods.
- You **cannot create objects of an interface unlike classes** which can be instantiated and used through its instance/object.
- Interfaces **do not have constructors** as they cannot be instantiated.
- Any **variable in an interface is public, static and final**. Hence, we need not specify these properties explicitly. Final variables are the one that can't be modified once they are initialized.
- Any **method in an interface is by default abstract and public**
- An **interface can extend only another interface** but not classes.
- A class **implementing an interface must provide implementation for all of its method** unless it's an abstract class.

Polymorphism

- A Dog IS-A Animal
- A Dog IS-A Mammal
- A Dog IS-A Carnivorous
- A Deer IS-A Dog
- A Deer IS-A Object


Object
Everything is an object!
Animal
(Silently extends Object)
Mammal
(extends Animal)
Dog
(extends Mammal)
Cat
(extends Mammal)
upcasting
downcasting
by Sinipull for codecall.net

Control Flows

if-then-else

```
void applyBrakes() {
    if (isMoving) {
        currentSpeed--;
    } else {
        System.err.println("The bicycle has already stopped!");
    }
}
```

```
if (testscore >= 90) {
    grade = 'A';
} else if (testscore >= 80) {
    grade = 'B';
} else if (testscore >= 70) {
    grade = 'C';
} else if (testscore >= 60) {
    grade = 'D';
} else {
    grade = 'F';
}
```

switch-case

```
switch (month) {
      case 1:  monthString = "January";
               break;
      case 2:  monthString = "February";
               break;
      case 3:  monthString = "March";
               break;
      case 4:  monthString = "April";
               break;
      case 12: monthString = "December";
               break;
      default: monthString = "Invalid month";
               break;
   }
```

do-while

```
class DoWhileDemo {
    public static void main(String[] args){
        int count = 1;
        do {
            System.out.println("Count is: " + count);
            count++;
        } while (count < 11);
    }
}
```

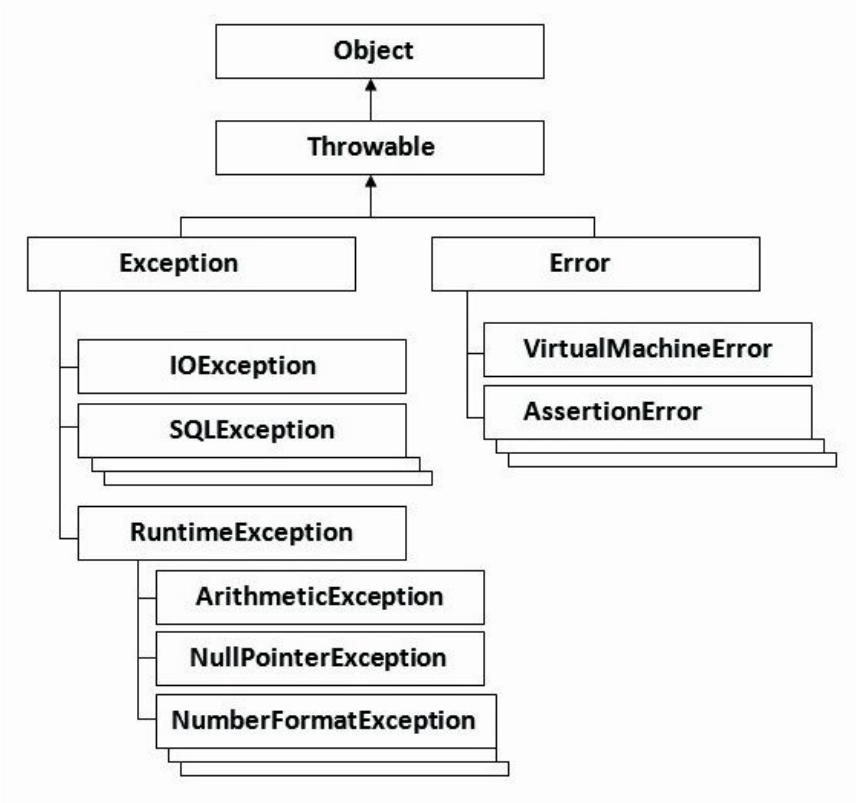
Exception Handling

Exception Handling in Java

The exception handling in java is one of the powerful **mechanism to handle the runtime errors and maintain the normal flow of application**.

Types of Exception

- Checked Exception
- Unchecked Exception
- Error



Exception Types

- **Checked Exception** : The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. **Checked exceptions are checked at compile-time**.
- **Unchecked Exception** : The **classes that extend RuntimeException** are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.
- **Error** : Error is **irrecoverable** e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Catching Exceptions

- If exception occurs in try block's body then control immediately transferred(**skipping rest of the statements in try block**) to the catch block. Once catch block finished execution then finally block and after that rest of the program.
- If there is no exception occurred in the code which is present in try block then first, the try block gets executed completely and then control gets transferred to finally block (**skipping catch blocks**).
- If a return statement is encountered either in try or catch block. In such case also **finally runs**. Control first goes to finally and then it returned back to **return statement**.

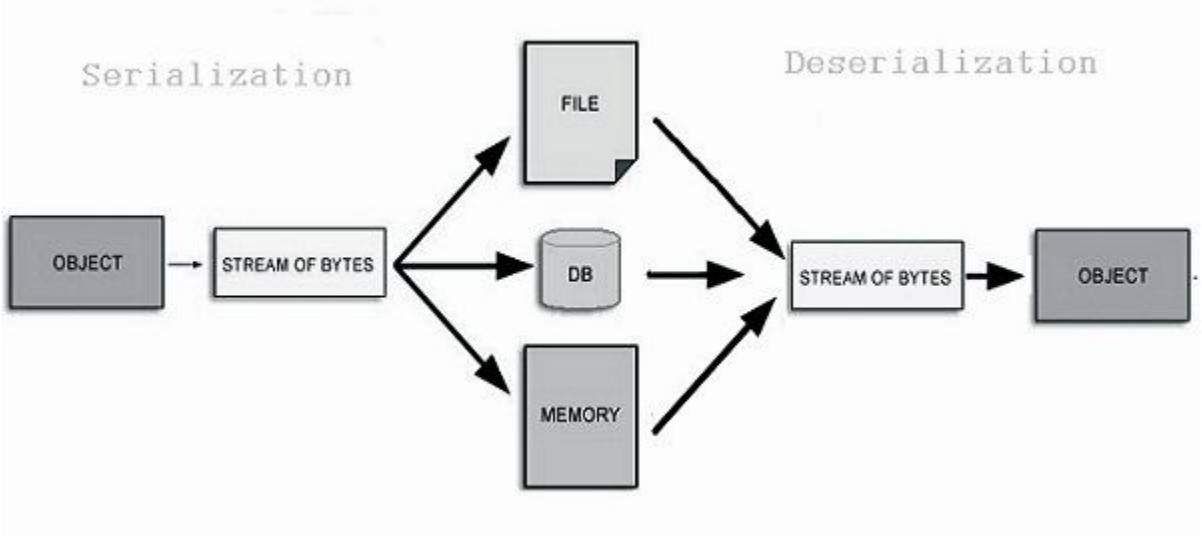
```
public class TestException{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(ArithmeticException e){System.out.println(e);}
  finally{
   System.out.println("finally block is always executed");}
   System.out.println("rest of the code...");
  }
}
```

Serialization and Deserialization

Serialization and Deserialization

Serialization is a process of **converting an object into a sequence of bytes** which can be persisted. The reverse process of **creating object from sequence of bytes** is called deserialization.

A **class must implement Serializable interface** present in java.io package in order to serialize its object successfully. Serializable is a marker interface that adds serializable behaviour to the class implementing it.




```
public final void writeObject(object x) throws IOException
```

```
public final Object readObject() throws IOException,ClassNotFoundException
```

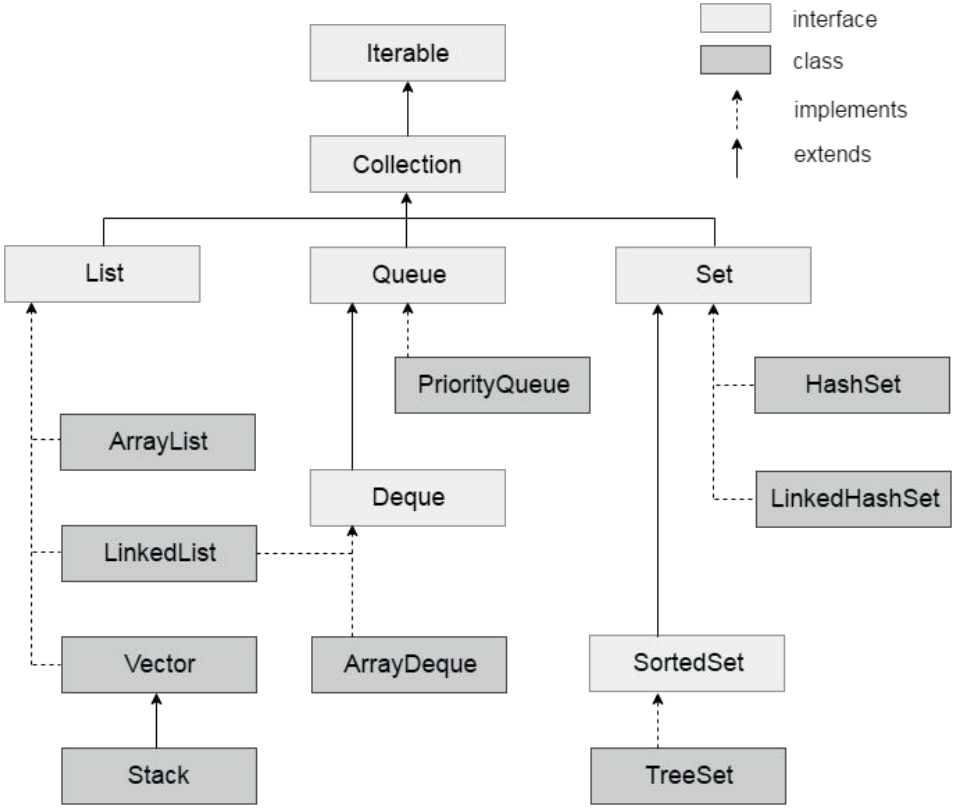
```
import java.io.*;
class Studentinfo implements Serializable {
    String name;
    int rid;
    static String contact;
    studentinfo(string n, int r, string c) {
        this.name = n;
        this.rid = r;
        this.contact = c;
    }
}

class Test {
    public static void main(String[] args) {
        try {
            Studentinfo si = new studentinfo("Abhi", 104, "110044");
            FileOutputStream fos = new
FileOutputStream("student.ser");
            Objectoutputstream oos = new ObjectOutputStream(fos);
            oos.writeObject(si);
            oos.close();
            fos.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
import java.io.*;
class DeserializationTest {
    public static void main(String[] args) {
        Studentinfo si = null;
        try {
            FileInputStream fis = new
FileInputStream("student.ser");
            ObjectOutputStream ois = new
ObjectOutputStream(fis);
            si = (Studentinfo) ois.readObject();
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println(si.name);
        System.out.println(si.rid);
        System.out.println(si.contact);
    }
}
```

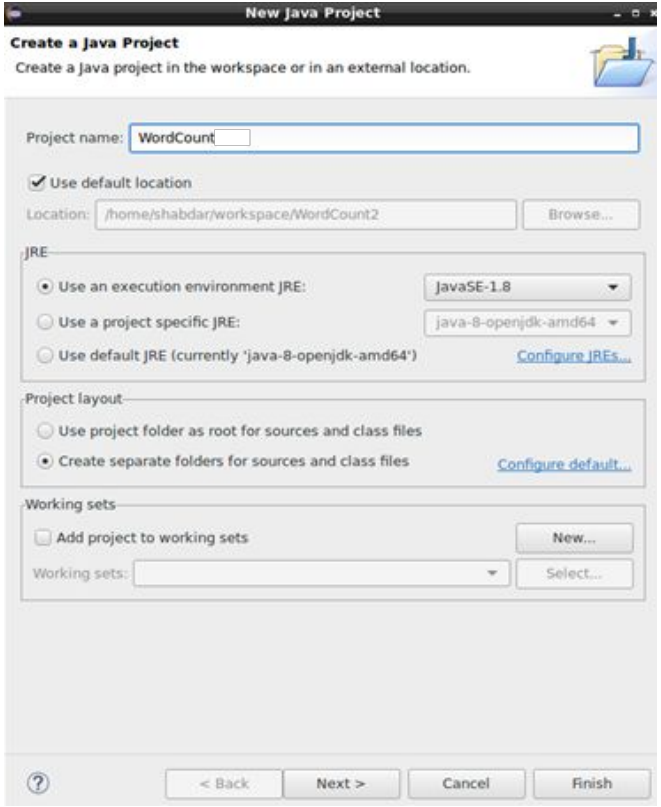
Collections

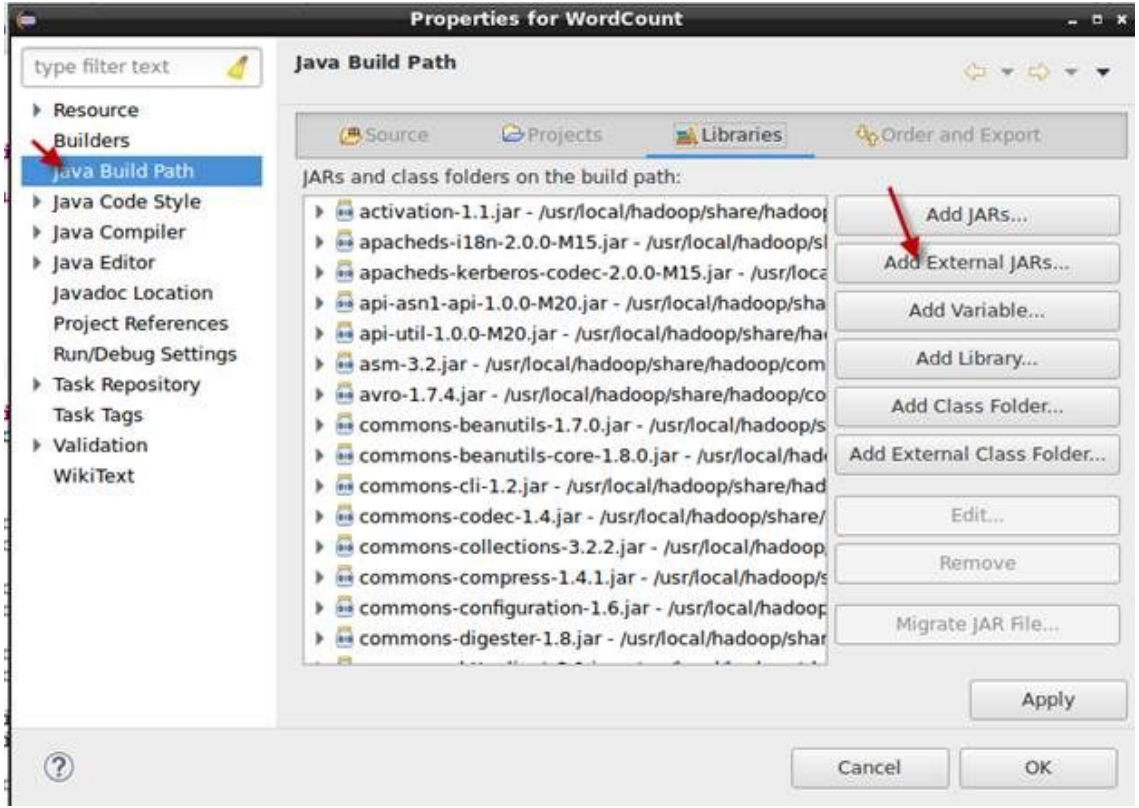
What are collections

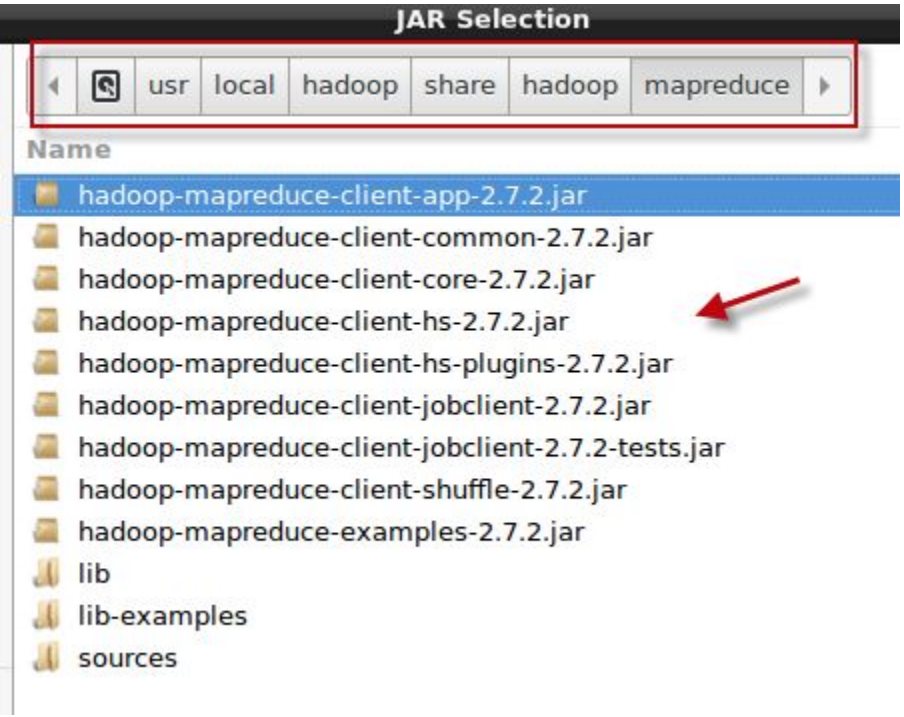
Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

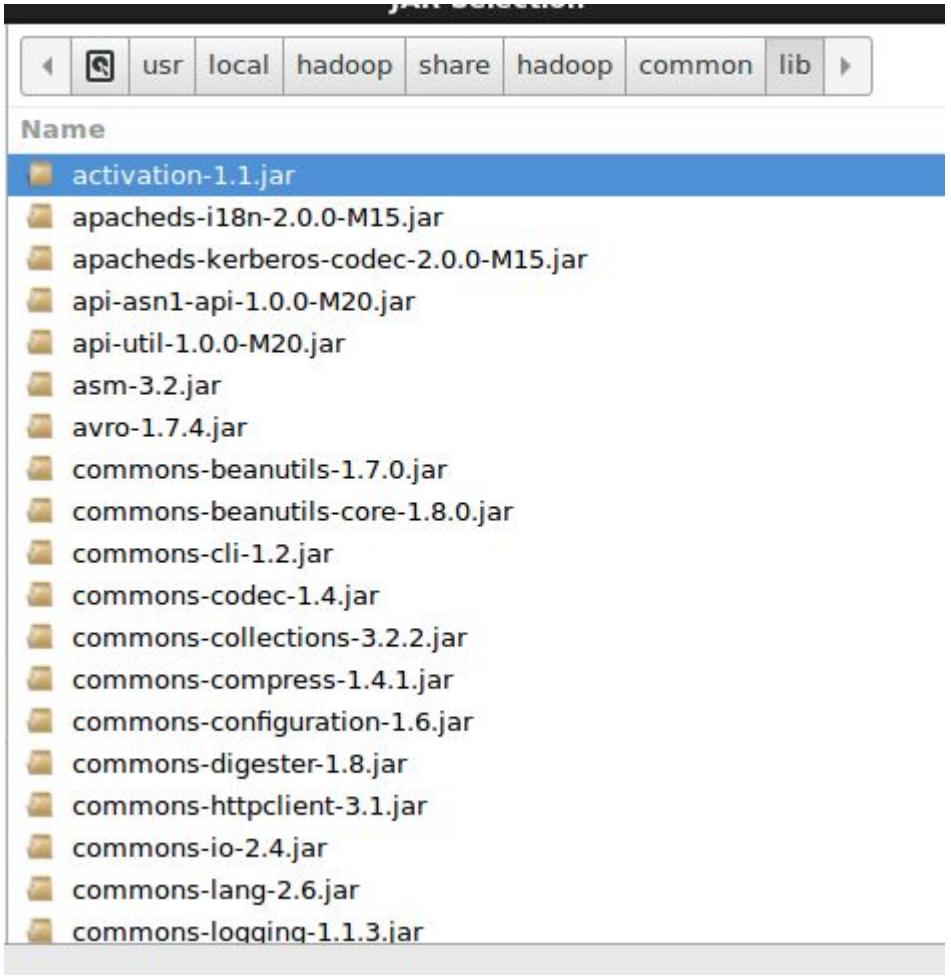


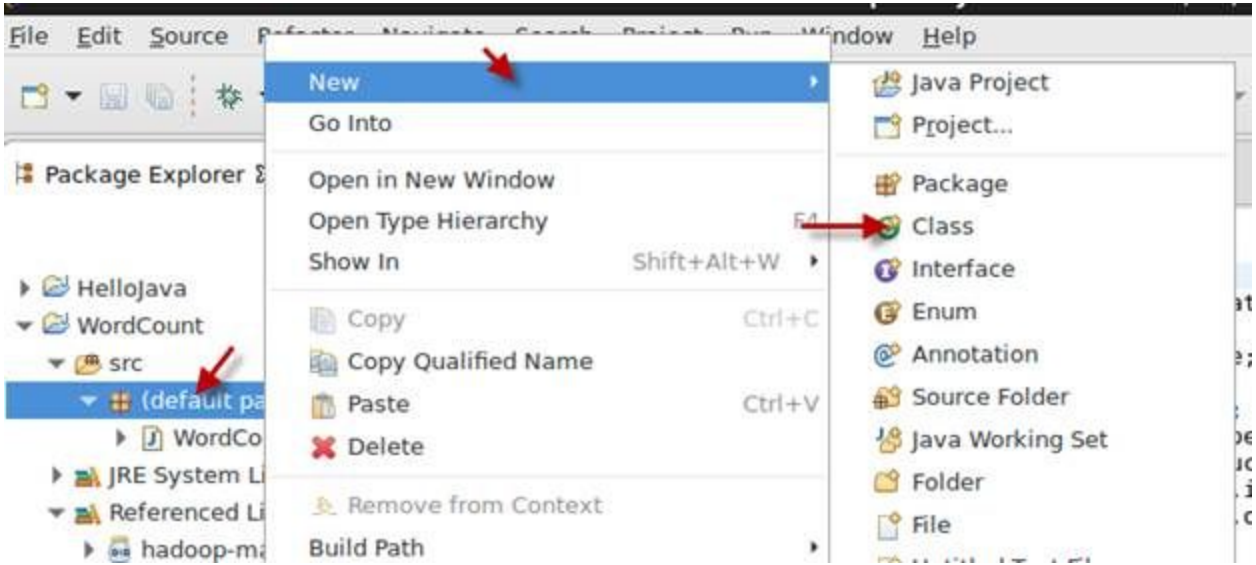
MapReduce using Java


New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
Project name: WordCount
Use default location
Location: /home/shabdar/workspace/WordCount2
Browse...
JRE
Use an execution environment JRE: JavaSE-1.8
Use a project specific JRE: java-8-openjdk-amd64
Use default JRE (currently 'java-8-openjdk-amd64')
Configure JREs...
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
New...
Working sets:
Select...
< Back
Next >
Cancel
Finish


Properties for WordCount
type filter text
Resource
Builders
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Project References
Run/Debug Settings
Task Repository
Task Tags
Validation
WikiText
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
activation-1.1.jar - /usr/local/hadoop/share/hadoop
apacheds-i18n-2.0.0-M15.jar - /usr/local/hadoop/sh
apacheds-kerberos-codec-2.0.0-M15.jar - /usr/loca
api-asn1-api-1.0.0-M20.jar - /usr/local/hadoop/sha
api-util-1.0.0-M20.jar - /usr/local/hadoop/share/ha
asm-3.2.jar - /usr/local/hadoop/share/hadoop/com
avro-1.7.4.jar - /usr/local/hadoop/share/hadoop/co
commons-beanutils-1.7.0.jar - /usr/local/hadoop/s
commons-beanutils-core-1.8.0.jar - /usr/local/had
commons-cli-1.2.jar - /usr/local/hadoop/share/had
commons-codec-1.4.jar - /usr/local/hadoop/share/
commons-collections-3.2.2.jar - /usr/local/hadoop
commons-compress-1.4.1.jar - /usr/local/hadoop/s
commons-configuration-1.6.jar - /usr/local/hadoop
commons-digester-1.8.jar - /usr/local/hadoop/shar
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
Apply
Cancel
OK

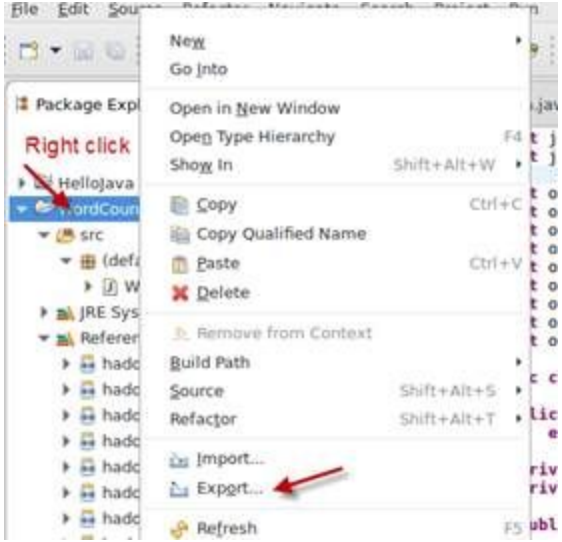

JAR Selection
usr
local
hadoop
share
hadoop
mapreduce
Name
hadoop-mapreduce-client-app-2.7.2.jar
hadoop-mapreduce-client-common-2.7.2.jar
hadoop-mapreduce-client-core-2.7.2.jar
hadoop-mapreduce-client-hs-2.7.2.jar
hadoop-mapreduce-client-hs-plugins-2.7.2.jar
hadoop-mapreduce-client-jobclient-2.7.2.jar
hadoop-mapreduce-client-jobclient-2.7.2-tests.jar
hadoop-mapreduce-client-shuffle-2.7.2.jar
hadoop-mapreduce-examples-2.7.2.jar
lib
lib-examples
sources

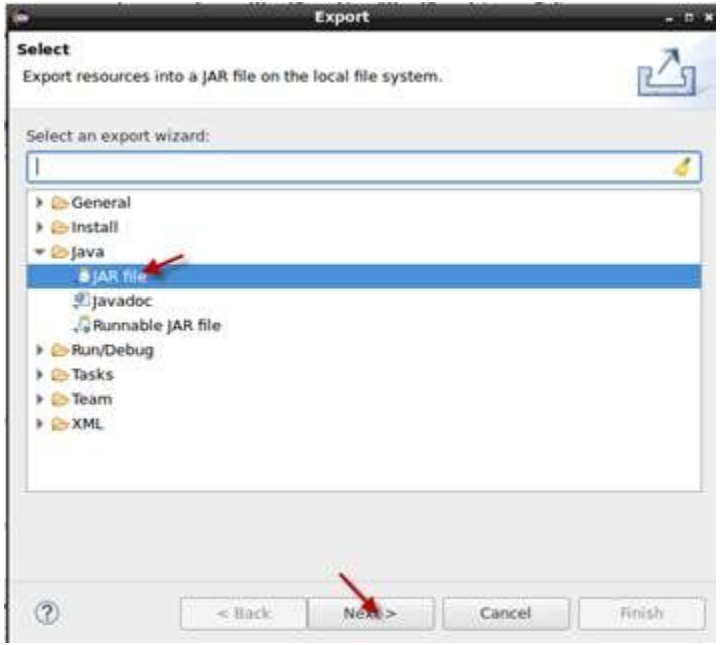

usr
local
hadoop
share
hadoop
common
lib
Name
activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
asm-3.2.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.1.3.jar

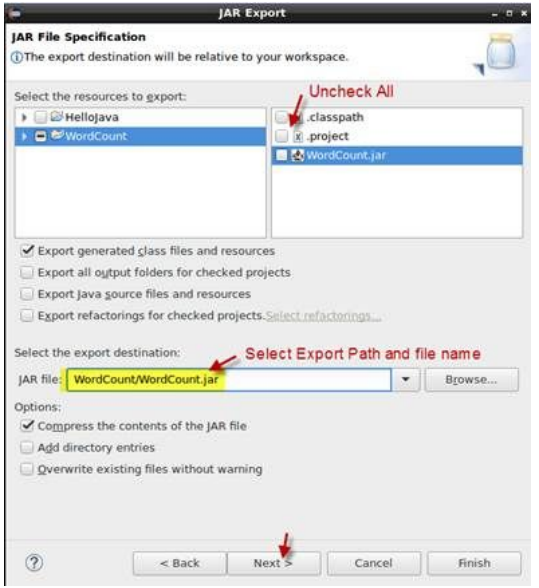

File
Edit
Source
Help
Package Explorer
HelloJava
WordCount
src
(default pa
WordCo
JRE System Li
Referenced Li
hadoop-ma
New
Go Into
Open in New Window
Open Type Hierarchy
F4
Show In
Shift+Alt+W
Copy
Ctrl+C
Copy Qualified Name
Paste
Ctrl+V
Delete
Remove from Context
Build Path
Java Project
Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Java Working Set
Folder
File

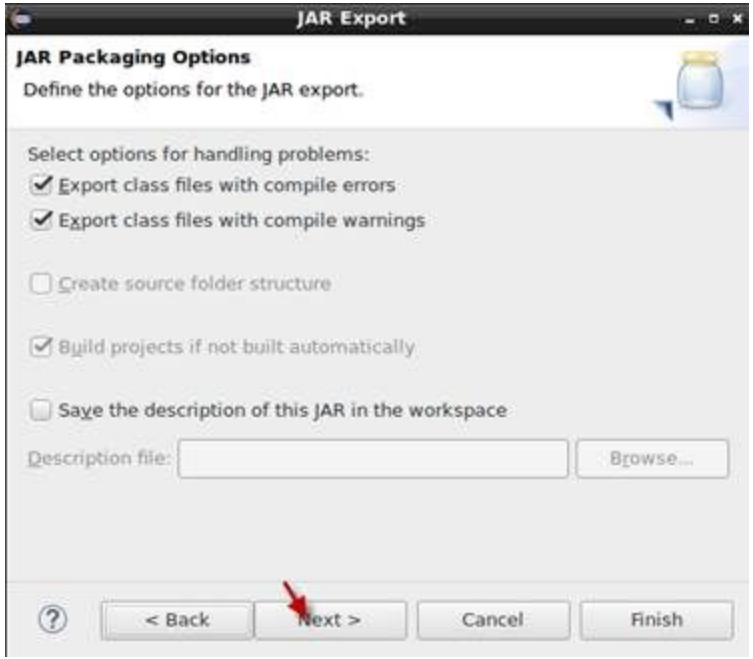
Implementation of a MapReduce program

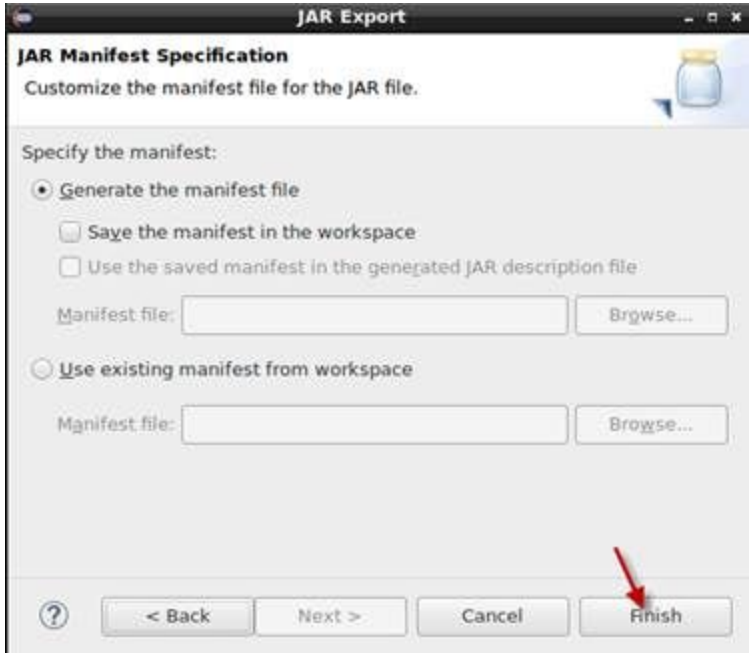
```
public class WordCount {

  public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{
    // ----
    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
      // ----
  }

  public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
   // ----
    public void reduce(Text key, Iterable<IntWritable> values,Context context) throws
IOException, InterruptedException {
      // ----
    }
  }

  public static void main(String[] args) throws Exception {
    // ----
  }
}
```


Right click
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Shift+Alt+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete
Remove from Context
Build Path
Source Shift+Alt+S
Refactor Shift+Alt+T
Import...
Export...
Refresh F5


Export
Select
Export resources into a JAR file on the local file system.
Select an export wizard:
General
Install
Java
JAR file
Javadoc
Runnable JAR file
Run/Debug
Tasks
Team
XML
< Back
Next >
Cancel
Finish


JAR Export
JAR File Specification
The export destination will be relative to your workspace.
Select the resources to export:
Uncheck All
HelloJava
WordCount
.classpath
.project
WordCount.jar
Export generated class files and resources
Export all output folders for checked projects
Export Java source files and resources
Export refactorings for checked projects. Select refactorings...
Select the export destination:
Select Export Path and file name
JAR file: WordCount/WordCount.jar
Browse...
Options:
Compress the contents of the JAR file
Add directory entries
Overwrite existing files without warning
< Back
Next >
Cancel
Finish


JAR Export
JAR Packaging Options
Define the options for the JAR export.
Select options for handling problems:
Export class files with compile errors
Export class files with compile warnings
Create source folder structure
Build projects if not built automatically
Save the description of this JAR in the workspace
Description file:
Browse...
< Back
Next >
Cancel
Finish


JAR Export
JAR Manifest Specification
Customize the manifest file for the JAR file.
Specify the manifest:
Generate the manifest file
Save the manifest in the workspace
Use the saved manifest in the generated JAR description file
Manifest file:
Browse...
Use existing manifest from workspace
Manifest file:
Browse...
< Back
Next >
Cancel
Finish

Once exported, it is time to run JAR file. But before that, create input folder on HDFS that will hold one or more text files for counting words.

$> hadoop fs -mkdir input

Create a text file with some text and copy it in Input folder. You can create multiple text files if you wish.

$> vi wordcounttest.txt

Add some sample test in this file and save it. Under Ubuntu you can also use nano editor which is easier than vi editor.

$> hadoop fs -put wordcounttest.txt input/wordcounttest.txt

Make sure output directory do not exist yet. If it already exists then remove it with below command, otherwise JAR will throw an error that directory already exist.

$> hadoop fs -rm -r output

Now run JAR file with command as below. Please note you may need to adjust path of jar file depending on where you exported it.

$> hadoop jar /home/rohit/workspace/WordCount/WordCount.jar WordCount input/ output/

Please note that this program will read all text files from input folder on HDFS and count total number of words in each file. Output is stored under output folder on HDFS.

Once run is successful, you should see below 2 files in output folder.

$> hadoop fs -ls output

```
-rw-r--r--   1 rohit rohit          0 2017-04-12 13:54 output/_SUCCESS
-rw-r--r--   1 rohit rohit         73 2017-04-12 13:54 output/part-r-00000
```

_SUCCESS file is an empty file indicating RUN was successful. To see output of run, view part-r-00000 file.

$> hdfs fs -cat output/part-r-00000

Linux

What is Linux?

Linux is **an operating system**.

An **operating system is a software that enables communication between computer hardware and software. It conveys input to get processed by the processor and brings output to the hardware to display it**. This is the basic function of an operating system.

Structure of Linux

- Kernel
- System Libraries
- System Tools

Kernel

Core of the operating system. It establishes communication between devices and software. Moreover, it manages the system resources.

- Device Management
- Memory Management
- Process Management
- Handling System Calls

Linux Commands

A command is an instruction given to our computer by us to do whatever we want. In Mac OS, and Linux it is called terminal, whereas, in windows it is called command prompt. Commands are always case sensitive.

Commands are executed by typing in at the command line followed by pressing enter key.

This command further passes to the shell which reads the command and execute it. Shell is a method for the user to interact with the system. Default shell in Linux is called bash (Bourne-Again Shell).

FILE COMMANDS

```
ls –al                          =>Display all information about files/ directories
pwd                             =>Show the path of current directory
mkdir directory-name            =>Create a directory
rm file-name                    =>Delete file
rm -r directory-nam             =>Delete directory recursively
rm -f file-name                 =>Forcefully remove file
rm -rf directory-name           =>Forcefully remove directory recursively
cp file1 file2                  =>Copy file1 to file2
cp -r dir1 dir2                 =>Copy dir1 to dir2, create dir2 if it doesn't exist
mv file1 file2                  =>Rename source to dest / move source to directory
ln –s /path/to/file-name link-name          #Create symbolic link to file-name
touch file                      =>Create or update file
cat > file                      =>Place standard input into file
more file                       =>Output contents of file
head file                       =>Output first 10 lines of file
tail file                       =>Output last 10 lines of file
tail -f file                    =>Output contents of file as it grows starting with the last
                                  10 lines
gpg -c file                     =>Encrypt file
gpg file.gpg                    =>Decrypt file
wc                              =>print the number of bytes, words, and lines in files
xargs                           =>Execute command lines from standard input
```

PROCESS RELATED

```
ps                          =>Display your currently active processes
ps aux | grep 'telnet'      =>Find all process id related to telnet process
pmap                        =>Memory map of process
top                         =>Display all running processes
kill pid                    =>Kill process with mentioned pid id
killall proc                =>Kill all processes named proc
pkill process-name          =>Send signal to a process with its name
bg                          =>Resumes suspended jobs without bringing them to
                              foreground
fg                          =>Brings the most recent job to foreground
fg n                        =>Brings job n to the foreground
```

DIRECTORY TRAVERSE

```
cd ..                =>To go up one level of the directory tree
cd                   =>Go to $HOME directory
cd /test             =>Change to /test directory
```

FILE TRANSFER

```
scp
scp file.txt server2:/tmp          =>Secure copy file.txt to remote host /tmp folder rsync
rsync -a /home/apps /backup/  =>Synchronize source to destination
```

DISK USAGE

```
df –h                =>Show free space on mounted filesystems
df -i                =>Show free inodes on mounted filesystems
fdisk -l             =>Show disks partitions sizes and types
du -ah               =>Display disk usage in human readable form
du -sh               =>Display total disk usage on the current directory
findmnt              =>Displays target mount point for all filesystem
mount device-path mount-point    =>Mount a device
```

COMPRESSION / ARCHIVES

```
tar cf home.tar home              =>Create tar named home.tar containing home/
tar xf file.tar                   =>Extract the files from file.tar
tar czf file.tar.gz files         =>Create a tar with gzip compression
gzip file                         =>Compress file and renames it to file.gz
```

SEARCH

```
grep pattern files                =>Search for pattern in files
grep -r pattern dir               =>Search recursively for pattern in dir
locate file                       =>Find all instances of file
find /home/tom -name 'index*'     =>Find files names that start with "index"
find /home -size +10000k          =>Find files larger than 10000k in /home
```

SYSTEM

```
uname -a              =>Displaylinux system information
uname -r              =>Display kernel release information
uptime                =>Show how long the system has been running + load
hostname              =>Show system host name
hostname -i           =>Display the IP address of the host
last reboot           =>Show system reboot history
date                  =>Show the current date and time
cal                   =>Show this month calendar
w                     =>Display who is online
whoami                =>Who you are logged in as
finger user           =>Display information about user
```

FILE PERMISSION RELATED

```
chmod octal file-name        =>Change the permissions of file to octal
Example
chmod 777 /data/test.c       =>Set rwx permission for owner,group,world
chmod 755 /data/test.c       =>Set rwx permission for owner,rx for group and world
chown owner-user file        =>Change owner of the file
chown owner-user:owner-group file-name          =>Change owner and group
                                                  owner of the file
chown owner-user:owner-group directory          =>Change owner and group
                                                  owner of the directory
```

Questions