1. Steps to install java

Step 1: Download JDK

Goto Java SE download site @ http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Under "Java Platform, Standard Edition" ⇒ "Java SE 8u{xx}", where {xx} is the latest update number ⇒ Click the "JDK Download" button.

Under "Java SE Development Kit 8u{xx}" ⇒ Check "Accept License Agreement".

Choose the JDK for your operating system, e.g., "Windows x64" (for 64-bit Windows OS) or "Windows x86" (for 32-bit Windows OS). You can check whether your Windows OS is 32-bit or 64-bit via "Control Panel" ⇒ (Optional) System and Security ⇒ System ⇒ Under "System Type".

Step 2: Install JDK and JRE

Step 3: Include JDK's "bin" Directory in the PATH

Windows OS searches the current directory and the directories listed in the PATH environment variable for executable programs. JDK's programs (such as Java compiler javac.exe and Java runtime java.exe) reside in directory "<JAVA\_HOME>\bin" (where <JAVA\_HOME> denotes the JDK installed directory). You need to include "<JAVA\_HOME>\bin" in the PATH to run the JDK programs.

To edit the PATH environment variable in Windows 7/8/10:

Launch "Control Panel" ⇒ (Optional) System and Security ⇒ System ⇒ Click "Advanced system settings" on the left pane.

Switch to "Advanced" tab ⇒ Push "Environment Variables" button.

Under "System Variables" (the bottom pane), scroll down to select "Path" ⇒ Click "Edit..."

Step 4: Verify the JDK Installation

Launch a CMD shell via one of the following means:

Click "Search" button ⇒ Enter "cmd" ⇒ Choose "Command Prompt", or

right-click "Start" button ⇒ run... ⇒ enter "cmd", or

(Prior to Windows 10) click "Start" button ⇒ All Programs ⇒ Accessories (or Windows System) ⇒ Command Prompt, or

(Windows 10) click "Start" button ⇒ Windows System ⇒ Command Prompt

Step 5: Write a Hello-World Java Program

Create a directory to keep your works, e.g., "d:\myProject", or "c:\myProject", or any directory of your choice. Do NOT save your works in "Desktop" or "Documents" as they are hard to locate. The directory name shall not contain blank or special characters. Use meaningful but short name as it is easier to type.

Launch a programming text editor (such as TextPad or NotePad++ or Sublime Text). Begin with a new file and enter the following source code. Save the file as "Hello.java", under your work directory (e.g., d:\myProject).

step 6: Compile and Run the Hello-World Java Program

1. Steps to install eclipse

Click Eclipse

it is critical that Java, Python, and Eclipse are either all 32 Bit or are all 64 Bit (and only if your Machine/OS supports 64 Bit): I think it easiest to use 32 Bit for everything.

Click the Windows 32 Bit Operating System for your machine, under the heading Eclipse Standard 4.4 (right under Package Solutions).

Click the orange DOWNLOAD button. The site named here, in orange to the right of the button: United States - Columbia University (http) is the random one chosen by the download page this time; yours may differ.

Terminate the window browsing the Eclipse download.

Move this file to a more permanent location, so that you can install Eclipse (and reinstall it later, if necessary).

Start the Installing instructions directly below.

1. Steps to create work space

Press the Next button and then Browse for the old projects you would like to import. Check "Copy projects into workspace" to make a copy. In Preferences->Workspaces, make sure that 'Prompt for Workspace on startup' is checked. Then you'll be prompted for a workspace to open

1. Steps to create project

Install the Eclipse IDE for Java Developers. ...

Click "File" → "New" → "Java Project". ...

Give the project a name. ...

Select the location for the project files. ...

Select Java Runtime Environment (JRE) you want to use. ...

Select your project layout. ...

Click "Next" to open the "Java Settings" window.

Use the Source tab to define your build path.

Use the Libraries tab to add libraries to the project.

Click "Finish" to start working on your new project.

1. Create java file/class

Ensure the source folder and package are correct.

Enter the class name.

Select the appropriate class modifier.

Enter the super class name or click on the Browse button to search for an existing class.

Click on the Add button to select the interfaces implemented by this class.

Examine and modify the check boxes related to method stubs and comments.

Click the Finish button.

1. What is main method will do?

All Java applications begin processing with a main() method;

Each statement in the main executes in order until the end of main is reached -- this is when your program terminates;

What does static mean? static means that you don't have to instantiate a class to call the method;

String[] args is an array of String objects. If you were to run your program on the command line, you could pass in parameters as arguments. These parameters can then be accessed as you would access elements in an array: args[0]...args[n];

public means that the method can be called by any object.

1. Creating property/data members?

Properties is a subclass of Hashtable. It is used to maintain lists of values in which the key is a String and the value is also a String.

The Properties class is used by many other Java classes. For example, it is the type of object returned by System.getProperties( ) when obtaining environmental values

public static void main(String args[]) {

Properties capitals = new Properties();

Set states;

String str;

1. What is data type and different data types?

There are two data types available in Java −

Primitive Data Types

Reference/Object Data Types

Primitive Data Types

There are eight primitive datatypes supported by Java. Primitive datatypes are predefined by the language and named by a keyword. Let us now look into the eight primitive data types in detail.

byte

Byte data type is an 8-bit signed two's complement integer

Minimum value is -128 (-2^7)

Maximum value is 127 (inclusive)(2^7 -1)

Default value is 0

Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.

Example: byte a = 100, byte b = -50

short

Short data type is a 16-bit signed two's complement integer

Minimum value is -32,768 (-2^15)

Maximum value is 32,767 (inclusive) (2^15 -1)

Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an integer

Default value is 0.

Example: short s = 10000, short r = -20000

int

Int data type is a 32-bit signed two's complement integer.

Minimum value is - 2,147,483,648 (-2^31)

Maximum value is 2,147,483,647(inclusive) (2^31 -1)

Integer is generally used as the default data type for integral values unless there is a concern about memory.

The default value is 0

Example: int a = 100000, int b = -200000

long

Long data type is a 64-bit signed two's complement integer

Minimum value is -9,223,372,036,854,775,808(-2^63)

Maximum value is 9,223,372,036,854,775,807 (inclusive)(2^63 -1)

This type is used when a wider range than int is needed

Default value is 0L

Example: long a = 100000L, long b = -200000L

float

Float data type is a single-precision 32-bit IEEE 754 floating point

Float is mainly used to save memory in large arrays of floating point numbers

Default value is 0.0f

Float data type is never used for precise values such as currency

Example: float f1 = 234.5f

double

double data type is a double-precision 64-bit IEEE 754 floating point

This data type is generally used as the default data type for decimal values, generally the default choice

Double data type should never be used for precise values such as currency

Default value is 0.0d

Example: double d1 = 123.4

boolean

boolean data type represents one bit of information

There are only two possible values: true and false

This data type is used for simple flags that track true/false conditions

Default value is false

Example: boolean one = true

char

char data type is a single 16-bit Unicode character

Minimum value is '\u0000' (or 0)

Maximum value is '\uffff' (or 65,535 inclusive)

Char data type is used to store any character

Example: char letterA = 'A'

Reference Datatypes

Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy, etc.

Class objects and various type of array variables come under reference datatype.

Default value of any reference variable is null.

A reference variable can be used to refer any object of the declared type or any compatible type.

Example: Animal animal = new Animal("giraffe");

1. What is variable?

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in the memory.

1. Creating method with void?

Considering the following example to explain the syntax of a method −

Syntax

public static int methodName(int a, int b) {

// body

}

Here,

public static − modifier

int − return type

methodName − name of the method

a, b − formal parameters

int a, int b − list of parameters

Method definition consists of a method header and a method body. The same is shown in the following syntax −

Syntax

modifier returnType nameOfMethod (Parameter List) {

// method body

}

The syntax shown above includes −

modifier − It defines the access type of the method and it is optional to use.

returnType − Method may return a value.

nameOfMethod − This is the method name. The method signature consists of the method name and the parameter list.

Parameter List − The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.

method body − The method body defines what the method does with the statements.

1. creating method with return data type, we can return int/string/double/float/date etc

tatic <T extends Number> T parseString(String str, Class<T> cls) {

if (cls == Float.class) {

return (T) Float.valueOf(str);

} else if (cls == Integer.class) {

return (T) Integer.valueOf(str);

}

1. method that will return hardcoded value

public final String value;

public final int index;

public ReturningValues(String value, int index) {

this.value = value;

this.index = index;

}

}

1. Create default/paramterzied constructors

A constructor with arguments is known as parameterized constructor.

Consider the below code: Here I have declared three constructors: one is default and two are parameterized. As you can see when we created the objects of class, these constructor got invoked, based on the parameters (or arguments) we provided during object creation.

class Example{

//Default constructor

Example(){

System.out.println("Default constructor");

}

/\* Parameterized constructor with

\* two integer arguments

\*/

Example(int i, int j){

System.out.print("parameterized constructor");

System.out.println(" with Two parameters");

}

/\* Parameterized constructor with

\* three integer arguments

\*/

Example(int i, int j, int k){

System.out.print("parameterized constructor");

System.out.println(" with Three parameters");

}

public static void main(String args[]){

//This will invoke default constructor

Example obj = new Example();

/\* This will invoke the constructor

\* with two arguments

\*/

Example obj2 = new Example(12, 12);

/\* This will invoke the constructor

\* with three arguments

\*/

Example obj3 = new Example(1, 2, 13);

}

}

1. method that will return property value

Add and remove a property named "color":

$("button").click(function(){

var $x = $("div");

$x.prop("color", "FF0000");

$x.append("The color property: " + $x.prop("color"));

$x.removeProp("color");

});

1. creating method with return data type and parameter

A method returns to the code that invoked it when it

* completes all the statements in the method,
* reaches a return statement, or
* throws an exception (covered later),

Any method that is not declared void must contain a return statement with a corresponding return value, like this:

return returnValue;

The data type of the return value must match the method's declared return type; you can't return an integer value from a method declared to return a boolean.

The getArea() method in the Rectangle Rectangle class that was discussed in the sections on objects returns an integer:

// a method for computing the area of the rectangle

public int getArea() {

return width \* height;

}

1. creating static property:

private static methods as the membeers that belong to the class and not of any instance of the class.

Static member of the class are called even before the object of that class are created.

Like the most common example, you want to know how many of the instance of the Dog class is created. To find out this we write the code as follows.

public class Dog

{

private static int count; //static member

public Dog()

{ count++;

}

public static void main(String a[])

{ Dog d=new Dog();

Dog dd=new Dog();

new Dog();

Dog.display();

}

private static void display()

{

System.out.println(count);

}

}

1. creating static method?

Static methods in java belong to the class (not an instance of it). They use no instance variables and will usually take input from the parameters, perform actions on it, then return some result. Instances methods are associated with objects and, as the name implies, can use instance variables.

public static int max3(int a, int b, int c) {

int max = a;

if (b > max) max = b;

if (c > max) max = c;

return max;

}

public static double max3(double a, double b, double c) {

double max = a;

if (b > max) max = b;

if (c > max) max = c;

return max;

}

1. create static block

* If you're loading drivers and other items into the namespace.
* If you need to do computation in order to initialize your static variables,you can declare a static block which gets executed exactly once,when the class is first loaded.
* Security related issues or logging related tasks.

static {

// whatever code is needed for initialization goes here

}

1. creating object

There are four different ways to create objects in java:

A. Using new keyword. This is the most common way to create an object in java. ...

B. Using Class.forName() If we know the name of the class & if it has a public default constructor we can create an object in this way.

C. Using clone() ...

D. Using object deserialization.

Point originOne = new Point(23, 94);

Rectangle rectOne = new Rectangle(originOne, 100, 200);

Rectangle rectTwo = new Rectangle(50, 100);

1. calling method with void

* A method is the equivalent of a function in languages like C which helps in code reusing. A set of statements make a method, and this method can be invoked through other statement. When invoked (called) , all the statements that are a part of the method would be executed. For instance, look at this method: "public static void methodExample() {}". It currently has no code in it, but there are three keywords before the method name. There is public, static, and void
* he word public before the method name means that the method itself can be called from anywhere which includes other classes, even from different packages (files) as long as you import the class. There are three other words that can replace public. They are protected and private. If a method is protected, then only this class and subclasses (classes that use this as a basis to build off of) can call the method. If a method is private, then the method can only be called inside the class. The last keyword is really not even a word. This is if you had nothing in the place of public, protected, or private. This is called the default, or package-private. This means that only the classes in the same package can call the method.
* The second keyword, static means that the method belongs to the class and not any instance of the class ( object ). Static methods must be called using the class name: "ExampleClass.methodExample()". However, if the keyword static was not there, then the method can be invoked only through an object. For instance, if the class was called ExampleObject and it had a constructor (for making objects), then we could make a new object by typing ExampleObject obj = new ExampleObject();, and call the method with "obj.methodExample();".
* The last word before the method name is void. The word void means that the method doesn't return anything (it does not return anything when you run the method). If you do want a method to return something, then simply replace the word void with a data type (primitive or reference type) of the object (or primitive type) that you wish to return. Then just add return plus an object of that type somewhere toward the end of the method's code.
* When calling a method that returns something, you can use what it returns. For example, if a someMethod() returns an integer, then you can set an integer to what it returns with "int a = someMethod();"
* Some methods require a parameter. A method that requires a parameter of an integer would look like someMethod(int a) When using a method like this, you would write the method name, and then an integer in the parentheses: someMethod(5) or someMethod(n) if n is an integer.
* Methods can also have multiple parameters, simply separated by commas. If the method someMethod required two parameters, int a and Object obj, it would look like "someMethod(int a, Object obj)". To use this new method, it would be called by the method name followed by an integer and an Object in parentheses: someMethod(4, thing) where thing is an Object.

1. calling method with no return and parameter

public static int methodName(int a, int b) {

// body

}

Here,

public static − modifier

int − return type

methodName − name of the method

a, b − formal parameters

int a, int b − list of parameters

Method definition consists of a method header and a method body. The same is shown in the following syntax −

Syntax

modifier returnType nameOfMethod (Parameter List) {

// method body

}

The syntax shown above includes −

modifier − It defines the access type of the method and it is optional to use.

returnType − Method may return a value.

nameOfMethod − This is the method name. The method signature consists of the method name and the parameter list.

Parameter List − The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.

method body − The method body defines what the method does with the statements.

1. calling method with return and no parameter

c lass Cell {

int i;

Cell c= new Cell(2);

Cell(Cell clone ) {

this.i = clone.i;

}

Cell(int i ) {

this.i = i;

}

public Cell division(){

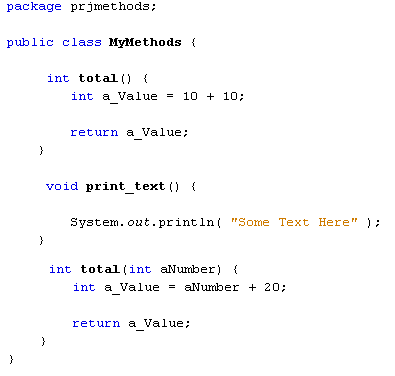
Cell tmpCell = new Cell(c);

return tmpCell;

}

}

1. calling method with return and parameter



1. calling static method

It is a method which belongs to the class and not to the object(instance)

A static method can access only static data. ...

A static method can call only other static methods and can not call a non-static method from it.

A static method can be accessed directly by the class name and doesn't need any object.

Syntax: <class-name>.<method-name>

1. create classes under multiple packages

package package1;

public class Package1Class {

}

package package2;

import package1.Package1Class;

public class Package2Class {

private Package1Class x;

public Package2Class (Package1Class x) {

this.x = x

}

}

1. calling classes under different packages

package jbt.access;

/\*

\* We will try to access the method(With DIfferent Access Modifiers) in this class from Other classes.

\*

\*/

public class JBT {

/\*

\* This method can be accessed from classes within the same package.

\*/

void defaultMethod() {

System.out.println("Inside Method with DEFAULT Access Modifier");

}

/\*

\* This method can be accessed from any class in Java world depend on the

\* visibility of CLASS.

\*/

public void publicMethod() {

System.out.println("Inside Method with PUBIC Access Modifier");

}

/\*

\* This method can not be accessed from outside of class.

\*/

private void privateMethod() {

System.out.println("Inside Method with PRIVATE Access Modifier");

}

/\*

\* This method can be accessed withing same package and subclass in any

\* package.

\*/

protected void protectedMethod() {

System.out.println("Inside Method with PROTECTED Modifier");

}

}

1. write code to handle exceptions with try/catch/finally

public void openFile(){

FileReader reader = null;

try {

reader = new FileReader("someFile");

int i=0;

while(i != -1){

i = reader.read();

System.out.println((char) i );

}

} catch (IOException e) {

//do something clever with the exception

} finally {

if(reader != null){

try {

reader.close();

} catch (IOException e) {

//do something clever with the exception

}

}

System.out.println("--- File End ---");

}

}

1. what is checked exception/unchecked exception

Checked: are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword

import java.io.\*;

class Main {

public static void main(String[] args) {

FileReader file = new FileReader("C:\\test\\a.txt");

BufferedReader fileInput = new BufferedReader(file);

// Print first 3 lines of file "C:\test\a.txt"

for (int counter = 0; counter < 3; counter++)

System.out.println(fileInput.readLine());

fileInput.close();

}

}

Unchecked are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.

class Main {

public static void main(String args[]) {

int x = 0;

int y = 10;

int z = y/x;

}

}

1. what is final keyword, create final class, final method, final property

class ChessAlgorithm {

enum ChessPlayer { WHITE, BLACK }

...

final ChessPlayer getFirstPlayer() {

return ChessPlayer.WHITE;

}

...

}

Methods called from constructors should generally be declared final. If a constructor calls a non-final method, a subclass may redefine that method with surprising or undesirable results

1. write code for interface and create class to implement that interface

an interface is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces. Extension is discussed later in this lesson.

Defining an interface is similar to creating a new class:

public interface OperateCar {

// constant declarations, if any

// method signatures

// An enum with values RIGHT, LEFT

int turn(Direction direction,

double radius,

double startSpeed,

double endSpeed);

int changeLanes(Direction direction,

double startSpeed,

double endSpeed);

int signalTurn(Direction direction,

boolean signalOn);

int getRadarFront(double distanceToCar,

double speedOfCar);

int getRadarRear(double distanceToCar,

double speedOfCar);

......

// more method signatures

}

Note that the method signatures have no braces and are terminated with a semicolon.

To use an interface, you write a class that implements the interface. When an instantiable class implements an interface, it provides a method body for each of the methods declared in the interface. For example,

public class OperateBMW760i implements OperateCar {

// the OperateCar method signatures, with implementation --

// for example:

int signalTurn(Direction direction, boolean signalOn) {

// code to turn BMW's LEFT turn indicator lights on

// code to turn BMW's LEFT turn indicator lights off

// code to turn BMW's RIGHT turn indicator lights on

// code to turn BMW's RIGHT turn indicator lights off

}

// other members, as needed -- for example, helper classes not

// visible to clients of the interface

}

1. implement method overloading

class DisplayOverloading

{

public void disp(char c)

{

System.out.println(c);

}

public void disp(char c, int num)

{

System.out.println(c + " "+num);

}

}

class Sample

{

public static void main(String args[])

{

DisplayOverloading obj = new DisplayOverloading();

obj.disp('a');

obj.disp('a',10);

}

}

1. implement method overriding

class Animal {

public void move() {

System.out.println("Animals can move");

}

}

class Dog extends Animal {

public void move() {

super.move(); // invokes the super class method

System.out.println("Dogs can walk and run");

}

}

public class TestDog {

public static void main(String args[]) {

Animal b = new Dog(); // Animal reference but Dog object

b.move(); // runs the method in Dog class

}

35) implementing polymorphism

package MethodOverride;

class Children //parent class

{

public void speak()//define method

{

System.out.println("Children speak in Hindi");

}

}

class Student extends Children//extend parent class

{

public void speak()//override method

{

System.out.println("Students can speak in English");

}

}

class Test

{

public static void main(String[] args)//main method

{ //create object of Children and Student class

Children c1 = new Children();

Student s = new Student();

//call method speak

c1.speak();

s.speak();

}

}

36) How to do inheritance in java (using extend keyword)

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

Syntax

class Super {

.....

.....

}

class Sub extends Super {

.....

.....

}

Code:

class Calculation {

int z;

public void addition(int x, int y) {

z = x + y;

System.out.println("The sum of the given numbers:"+z);

}

public void Subtraction(int x, int y) {

z = x - y;

System.out.println("The difference between the given numbers:"+z);

}

}

public class My\_Calculation extends Calculation {

public void multiplication(int x, int y) {

z = x \* y;

System.out.println("The product of the given numbers:"+z);

}

public static void main(String args[]) {

int a = 20, b = 10;

My\_Calculation demo = new My\_Calculation();

demo.addition(a, b);

demo.Subtraction(a, b);

demo.multiplication(a, b);

}

37) write code to retrieve items from integer, string array

/\*\*

\* Demonstrates several Java array examples, including a

\* Java int array, and a Java String array.

\* Created by Alvin Alexander, http://alvinalexander.com.

\*/

public class JavaIntArrayExample

{

public static void main(String[] args)

{

new JavaArrayExample();

}

public JavaArrayExample()

{

intArrayExample();

stringArrayExample();

intArrayExample2();

}

/\*\*

\* Create an int array, then populate the array,

\* and finally print each element in the int array.

\*/

private void intArrayExample()

{

int[] intArray = new int[3];

intArray[0] = 1;

intArray[1] = 2;

intArray[2] = 3;

System.out.println("intArray output");

for (int i=0; i<intArray.length; i++)

{

System.out.println(intArray[i]);

}

}

/\*\*

\* Create a String array, then populate the array,

\* and finally print each element in the int array.

\*/

private void stringArrayExample()

{

String[] stringArray = new String[3];

stringArray[0] = "a";

stringArray[1] = "b";

stringArray[2] = "c";

System.out.println("stringArray output");

for (int i=0; i<stringArray.length; i++)

{

System.out.println(stringArray[i]);

}

}

/\*\*

\* Create a Java int array and populate it in one step.

\* Then get the array length and print each element in the array.

\*/

private void intArrayExample2()

{

int[] intArray = new int[] {4,5,6,7,8};

System.out.println("intArray output (version 2)");

for (int i=0; i<intArray.length; i++)

{

System.out.println(intArray[i]);

}

}

}

38) write code to add items to ArrayList collection

import java.util.Collections;

import java.util.ArrayList;

public class Program {

public static void main(String[] args) {

ArrayList<String> list = new ArrayList<>();

list.add("cat");

list.add("bird");

list.add("ant");

list.add("dog");

// Sort the elements alphabetically.

Collections.sort(list);

for (String value : list) {

System.out.println(value);

}

}

}

39) write code to add items HashMap

package bindu;

import java.util.HashMap;

import java.util.Map;

import java.util.Iterator;

import java.util.Set;

public class Details {

public static void main(String args[]) {

/\* This is how to declare HashMap \*/

HashMap<Integer, String> hmap = new HashMap<Integer, String>();

/\*Adding elements to HashMap\*/

hmap.put(12, "Chaitanya");

hmap.put(2, "Rahul");

hmap.put(7, "Singh");

hmap.put(49, "Ajeet");

hmap.put(3, "Anuj");

/\* Display content using Iterator\*/

Set set = hmap.entrySet();

Iterator iterator = set.iterator();

while(iterator.hasNext()) {

Map.Entry mentry = (Map.Entry)iterator.next();

System.out.print("key is: "+ mentry.getKey() + " & Value is: ");

System.out.println(mentry.getValue());

}

/\* Get values based on key\*/

String var= hmap.get(2);

System.out.println("Value at index 2 is: "+var);

/\* Remove values based on key\*/

hmap.remove(3);

System.out.println("Map key and values after removal:");

Set set2 = hmap.entrySet();

Iterator iterator2 = set2.iterator();

while(iterator2.hasNext()) {

Map.Entry mentry2 = (Map.Entry)iterator2.next();

System.out.print("Key is: "+mentry2.getKey() + " & Value is: ");

System.out.println(mentry2.getValue());

}

}

}

40) write code to retrieve items HashMap

HashMap<String, String> facilities = new HashMap<String, String>();

Iterator i = facilities.entrySet().iterator();

for(Entry<String, String> entry : facilities.entrySet()) {

String key = entry.getKey();

String value = entry.getValue();

System.out.println(key + " " + value);

}41) Write code to add items to hashset

HashSet extends AbstractSet and implements the Set interface. It creates a collection that uses a hash table for storage.

A hash table stores information by using a mechanism called hashing. In hashing, the informational content of a key is used to determine a unique value, called its hash code.

import java.util.\*;

public class HashSetDemo {

public static void main(String args[]) {

// create a hash set

HashSet hs = new HashSet();

// add elements to the hash set

hs.add("B");

hs.add("A");

hs.add("D");

hs.add("E");

hs.add("C");

hs.add("F");

System.out.println(hs);

}

}

42) Write code to retrieve items to hashset

MyObject findIfPresent(MyObject source, HashSet<MyObject> set)

{

if (set.contains(source)) {

for (MyObject obj : set) {

if (obj.equals(source))

return obj;

}

}

return null;

}

43) write code to connect to JDBC to get rows from employee table

package test;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

//import com.mysql.jdbc.Driver;

public class MySqlJdbcTest {

public static void main(String[] args) {

Connection conn = null;

Statement stmt = null;

ResultSet rs = null;

try {

// new com.mysql.jdbc.Driver();

Class.forName("com.mysql.jdbc.Driver").newInstance();

// conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/testdatabase?user=testuser&password=testpassword");

String connectionUrl = "jdbc:mysql://localhost:3306/testdatabase";

String connectionUser = "testuser";

String connectionPassword = "testpassword";

conn = DriverManager.getConnection(connectionUrl, connectionUser, connectionPassword);

stmt = conn.createStatement();

rs = stmt.executeQuery("SELECT \* FROM employees");

while (rs.next()) {

String id = rs.getString("id");

String firstName = rs.getString("first\_name");

String lastName = rs.getString("last\_name");

System.out.println("ID: " + id + ", First Name: " + firstName

+ ", Last Name: " + lastName);

}

} catch (Exception e) {

e.printStackTrace();

} finally {

try { if (rs != null) rs.close(); } catch (SQLException e) { e.printStackTrace(); }

try { if (stmt != null) stmt.close(); } catch (SQLException e) { e.printStackTrace(); }

try { if (conn != null) conn.close(); } catch (SQLException e) { e.printStackTrace(); }

}

}

}

44) Difference between string, string buffer, string builder with example

String is immutable, if you try to alter their values, another object gets created, whereas StringBuffer and StringBuilder are mutable so they can change their values

String

String demo = " hello " ;

// The above object is stored in constant string pool and its value can not be modified.

demo="Bye" ; //new "Bye" string is created in constant pool and referenced by the demo variable

// "hello" string still exists in string constant pool and its value is not overrided but we lost reference to the "hello"string

String buffer

StringBuffer demo1 = new StringBuffer("Hello") ;

// The above object stored in heap and its value can be changed .

demo1=new StringBuffer("Bye");

// Above statement is right as it modifies the value which is allowed in the StringBuffer

String builder

StringBuilder demo2= new StringBuilder("Hello");

// The above object too is stored in the heap and its value can be modified

demo2=new StringBuilder("Bye");

// Above statement is right as it modifies the value which is allowed in the StringBuilder