Assignment 2-3

Qno.1) Types of inner classes in JAVA:

1. Nested Inner Class:

Purpose: Nested classes are used when multiple classes that are similar are grouped together to make the code more readable. It is useful if the classes are not to be used outer class

.

When to use: When multiple similar classes are used, it is always more convenient to use nested classes.

Limitations:A nested class cannot refer directly to instance variables or methods defined in its enclosing class.

Code:

public class Outer\_class{

class inner\_class{

public void print(){  
 System.out.println(“This is a method of a inner class”);

}

}

public static void main(String args[]){  
 inner\_class n1=new inner\_class();

n1.print();

}

}

1. Method local inner class:

Purpose:The scope of the inner class is restricted within the method.

When to use: When a certain class is only used in a method, we use method local inner classes.

Limitations:A method-local inner class can be instantiated only within the method where the inner class is defined.

Code:

class OuterClass{

public void method(){  
 class InnerClass{

public void InnerMethod(){  
 System.out.println(“method in inner class”);

}

InnerClass obj = new InnerClass();

obj.InnerMethod();

}

}

public static void main(String args[]){

OuterClass obj=new OuterClass();

obj.method();

}

}

1. Anonymous Inner Class:

Purpose: Anonymous classes enable you to make your code more concise. They enable you to declare and instantiate a class at the same time.

When to use: When a local class is to be used only once,

Anonymous inner classes are preferred.

Limitations:An anonymous class cannot access local variables in its enclosing scope that are not declared as final.

Code:

public class OuterClass {

void print() {

System.out.println("I am in the print method of superclass");

}

class AnonymousClass {

static OuterClass out = new OuterClass() {

void print() {

super.print();

System.out.println("I am in Anonymous class");

}

}

};

public static void main(String[] args) {

out.print();

}

}

1. Static Nested Inner Classes:

Purpose:A static class that is created inside a class, is called a static nested class in Java. It is useful if the static classes are not to be used outer class

When to use: When the static class is of similar type of the outer class, it is nested.

Limitations:A static nested class cannot refer directly to instance variables or methods defined in its enclosing class.

Code:

class OuterClass {

public static class InnerClass {

public void print() {

System.out.println("I am printing from a static inner class!");

}

}

}

public class StaticInnerClass {

public static void main(String[] args) {

OuterClass.InnerClass in =new OuterClass.InnerClass(); in .print();

}

}

Q2.)

Q2) For reading and writing .bin files:

import java.io.BufferedInputStream;

import java.io.BufferedOutputStream;

import java.io.IOException;

import java.io.FileInputStream;

import java.io.FileOutputStream;

public class Q2 {

public static void main (String[] args) throws IOException {

String file="k201234.bin";

BufferedOutputStream writingOutput=new BufferedOutputStream(new FileOutputStream(file));

writingOutput.write(289);

writingOutput.write(327);

writingOutput.write(1234);

writingOutput.close();

System.out.println("Writing TASK COMPLETE!");

BufferedInputStream readingInput=new BufferedInputStream(new FileInputStream(file));

System.out.println(readingInput.read());

System.out.println(readingInput.read());

System.out.println(readingInput.read());

System.out.println("Reading and Displaying TASK COMPLETE!");

writingOutput.close();

}

}

For reading and writing .txt file:

import java.io.\*;

public class Q2 {

public static void main(String [] args) {

String fileinpc = "k201234.txt";

String line = null;

try {

FileReader fileReader =

new FileReader(fileinpc);

BufferedReader bufferedReader =

new BufferedReader(fileReader);

while((line = bufferedReader.readLine()) != null) {

System.out.println(line);

}

bufferedReader.close();

}

catch(FileNotFoundException ex) {

System.out.println("Unable to open file '" + fileinpc);

}

catch(IOException ex) {

System.out.println( "Error reading file '" + fileinpc);

}

}

}