Assignment 1

Submitted by: Kushagra Shukla

1900520100029

CSE III Year Student – IET Lucknow

1.

The reason behind this is to prevent ambiguity.

Consider a case where class B extends class A and Class C and both class A and C have the same method display (). Now java compiler cannot decide, which display method it should inherit. To prevent such situation, multiple inheritances is not allowed in java.

2.

* JDK is a software development kit whereas JRE is a software bundle that allows Java program to run, whereas JVM is an environment for executing bytecode.
* The full form of JDK is Java Development Kit, while the full form of JRE is Java Runtime Environment, while the full form of JVM is Java Virtual Machine.
* JDK is platform dependent, JRE is also platform dependent, but JVM is not platform independent.
* JDK contains tools for developing, debugging, etc. JRE contains class libraries and other supporting files, whereas software development tools are not included in JVM.

3.

 The idea is that the Java language is portable (or, more precisely, the compiled byte code is portable). You are sure that each VM requires a specific implementation for a certain hardware profile. However, once this effort has been made, the entire Java bytecode will be executed on this platform.

· The Java source code is compiled in bytecode when the javac compiler is used. The bytecode is saved on the disk with the file extension .class.

· Java is a compiled programming language, but instead of compiling directly to an executable machine code, it is compiled into an intermediate, binary format called JVM byte code. The byte code is compiled and / or interpreted to execute the program.

4.

This handler catches exceptions of type Exception; therefore, it catches any exception. This can be a poor implementation because you are losing valuable information about the type of exception being thrown and making your code less efficient. As a result, your program may be forced to determine the type of exception before it can decide on the best recovery strategy.

5.

1. Overriding implements Runtime Polymorphism whereas Overloading implements Compile time polymorphism.
2. The method Overriding occurs between superclass and subclass. Overloading occurs between the methods in the same class.
3. Overriding methods have the same signature i.e., same name and method arguments. Overloaded method names are the same but the parameters are different.
4. With Overloading, the method to call is determined at the compile-time. With overriding, the method call is determined at the runtime based on the object type.
5. If overriding breaks, it can cause serious issues in our program because the effect will be visible at runtime. Whereas if overloading breaks, the compile-time error will come and it’s easy to fix.

6.

public static void cat(File file) {

RandomAccessFile input = null;

String line = null;

try {

input = new RandomAccessFile(file, "r");

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

System.out.println(line);

}

return;

**} catch(FileNotFoundException fnf) {**

**System.err.format("File: %s not found%n", file);**

**} catch(IOException e) {**

**System.err.println(e.toString());**

} finally {

if (input != null) {

**try {**

input.close();

**} catch(IOException io) {**

**}**

}

}

}