Assignment

1. What are generics?

1. **import** java.util.\*;
2. **class** TestGenerics1{
3. **public** **static** **void** main(String args[]){
4. ArrayList<String> list=**new** ArrayList<String>();
5. list.add("rahul");
6. list.add("jai");
7. //list.add(32);//compile time error
9. String s=list.get(1);//type casting is not required
10. System.out.println("element is: "+s);
12. Iterator<String> itr=list.iterator();
13. **while**(itr.hasNext()){
14. System.out.println(itr.next());
15. }
16. }
17. }

2. Can we change the scope of the overridden method in the subclass for private, public,   
default and protected? Explain how can it be changed for each scope?

class A {

   protectedvoid protectedMethod() {

      System.out.println("superclass protected method");

   }

}

class B extends A {

   protected void protectedMethod() {

      System.out.println("subclass protected method");

   }

}

public class Test {

   public static void main(String args[]) {

      B b = new B();

      b.protectedMethod();

   }

}

3. What is the covariant return type?

1. **class** A{
2. A get(){**return** **this**;}
3. }
5. **class** B1 **extends** A{
6. @Override
7. B1 get(){**return** **this**;}
8. **void** message(){System.out.println("welcome to covariant return type");}
10. **public** **static** **void** main(String args[]){
11. **new** B1().get().message();
12. }
13. }

4. Can we override the static and private methods? Why?

class Base {

private static void display() {

System.out.println("Static or class method from Base");

}

public void print() {

System.out.println("Non-static or instance method from Base");

}

class Derived extends Base {

private static void display() {

System.out.println("Static or class method from Derived");

}

public void print() {

System.out.println("Non-static or instance method from Derived");

}

public class test {

public static void main(String args[])

{

Base obj= new Derived();

obj1.display();

obj1.print();

}

}

5. Difference between String Buffer and StringBuilder?

**package** edureka;

**public** **class** example {

**public** **static** **void** StrConcat(String str1)

    {

        str1 = str1 + "Edureka";

    }

**public** **static** **void** StrBufConcat(StringBuffer str2)

    {

        str2.append("Edureka");

    }

**public** **static** **void** StrBuildConcat(StringBuilder str3)

    {

        str3.append("Edureka");

    }

**public** **static** **void** main(String[] args)

    {

        String str1 = "Hello!";

        StrConcat(str1);

        System.out.println("The final String is - " + str1);

        StringBuffer str2 = **new** StringBuffer("Hello!");

        StrBufConcat(str2);

        System.out.println("The final String is - " + str2);

        StringBuilder str3 = **new** StringBuilder("Hello!");

        StrBuildConcat(str3);

        System.out.println("The final String is -" + str3);

    }

}

6. Difference between String class and String Buffer?

1. **public** **class** InstanceTest{
2. **public** **static** **void** main(String args[]){
3. System.out.println("Hashcode test of String:");
4. String str="java";
5. System.out.println(str.hashCode());
6. str=str+"tpoint";
7. System.out.println(str.hashCode());
9. System.out.println("Hashcode test of StringBuffer:");
10. StringBuffer sb=**new** StringBuffer("java");
11. System.out.println(sb.hashCode());
12. sb.append("tpoint");
13. System.out.println(sb.hashCode());
14. }
15. }

7. Can we declare constructor as final?

**import** java.io.\*;

**class** GFG {

    // GFG() constructor is declared final

**final** GFG()

    {

        // This line can not be executed as compile error

        // will come

        System.out.print(

            "Hey you have declared constructor as final, it's error");

    }

}

**class** Main {

**public** **static** **void** main(String[] args)

    {

        // Object of GFG class created

        // Automatically GFG() constructor called

        GFG obj = **new** GFG();

    }

}

8. Can we have try without catch block in java?

public class TryBlockWithoutCatch {

   public static void main(String[] args) {

      try {

         System.out.println("Try Block");

      } finally {

         System.out.println("Finally Block");

      }

   }

}

9. What is try with the resource?

1. **import** java.io.FileOutputStream;
2. **public** **class** TryWithResources {
3. **public** **static** **void** main(String args[]){
4. // Using try-with-resources
5. **try**(FileOutputStream fileOutputStream =newFileOutputStream("/java7-new-features/src/abc.txt")){
6. String msg = "Welcome to javaTpoint!";
7. **byte** byteArray[] = msg.getBytes(); //converting string into byte array
8. fileOutputStream.write(byteArray);
9. System.out.println("Message written to file successfuly!");
10. }**catch**(Exception exception){
11. System.out.println(exception);
12. }
13. }
14. }

10. Can we modify the throws clause of the superclass method while overriding it in the   
subclass?

**import** java.io.\*;

**class** SuperClass {

  // SuperClass doesn't declare any exception

**void** method() {

    System.out.println("SuperClass");

  }

}

// SuperClass inherited by the SubClass

**class** SubClass **extends** SuperClass {

  // method() declaring Checked Exception IOException

**void** method() **throws** IOException {

    // IOException is of type Checked Exception

    // so the compiler will give Error

    System.out.println("SubClass");

  }

  // Driver code

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

    SuperClass s = **new** SubClass();

    s.method();

  }

}

11. What is an association, aggregation, and composition in UML?

**import** java.io.\*;

// Class 1

// Engine class which will

// be used by car. so 'Car'

// class will have a field

// of Engine type.

**class** Engine {

    // Method to starting an engine

**public** **void** work()

    {

        // Print statement whenever this method is called

        System.out.println(

            "Engine of car has been started ");

    }

}

// Class 2

// Engine class

**final** **class** Car {

    // For a car to move,

    // it needs to have an engine.

    // Composition

**private** **final** Engine engine;

    // Note: Uncommented part refers to Aggregation

    // private Engine engine;

    // Constructor of this class

    Car(Engine engine)

    {

        // This keywords refers to same instance

**this**.engine = engine;

    }

    // Method

    // Car start moving by starting engine

**public** **void** move()

    {

        // if(engine != null)

        {

            // Calling method for working of engine

            engine.work();

            // Print statement

            System.out.println("Car is moving ");

        }

    }

}

// Class 3

// Main class

**class** GFG {

    // Main driver method

**public** **static** **void** main(String[] args)

    {

        // Making an engine by creating

        // an instance of Engine class.

        Engine engine = **new** Engine();

        // Making a car with engine so we are

        // passing a engine instance as an argument

        // while creating instance of Car

        Car car = **new** Car(engine);

        // Making car to move by calling

        // move() method inside main()

        car.move();

    }

}

12. Difference between final, finally and finalize()?

private final String halo = "Hello World!";

try {

//code here

} catch (SomeException se) {

//handle exception here

} finally {

//always executed this code block

}

protected void finalize() {

//free resources here

super.finalize();

}

13. Difference between Vector and ArrayList?

**import** java.io.\*;

**import** java.util.\*;

**class** GFG

{

**public** **static** **void** main (String[] args)

    {

        // creating an ArrayList

        ArrayList<String> al = **new** ArrayList<String>();

        // adding object to arraylist

        al.add("Practice.GeeksforGeeks.org");

        al.add("quiz.GeeksforGeeks.org");

        al.add("code.GeeksforGeeks.org");

        al.add("contribute.GeeksforGeeks.org");

        // traversing elements using Iterator'

        System.out.println("ArrayList elements are:");

        Iterator it = al.iterator();

**while** (it.hasNext())

            System.out.println(it.next());

        // creating Vector

        Vector<String> v = **new** Vector<String>();

        v.addElement("Practice");

        v.addElement("quiz");

        v.addElement("code");

        // traversing elements using Enumeration

        System.out.println("\nVector elements are:");

        Enumeration e = v.elements();

**while** (e.hasMoreElements())

            System.out.println(e.nextElement());

    }

}

14. What are the different ways to make ArrayList methods synchronized?

import java.util.\*;

public class Example {

   public static void main (String[] args) {

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

      list.add("Hello");

      list.add("Hi");

      list.add("World");

      list = Collections.synchronizedList(list);

      synchronized(list) {

         Iterator itr = list.iterator();

         while (itr.hasNext())

         System.out.print(itr.next()+" ");

      }

   }

}

15. Difference between Hash table and Hash Map?

**import** java.util.\*;

**import** java.lang.\*;

**import** java.io.\*;

// Name of the class has to be "Main"

// only if the class is public

**class** Ideone

{

**public** **static** **void** main(String args[])

    {

        //----------hashtable -------------------------

        Hashtable<Integer,String> ht=**new** Hashtable<Integer,String>();

        ht.put(101," ajay");

        ht.put(101,"Vijay");

        ht.put(102,"Ravi");

        ht.put(103,"Rahul");

        System.out.println("-------------Hash table--------------");

**for** (Map.Entry m:ht.entrySet()) {

            System.out.println(m.getKey()+" "+m.getValue());

        }

        //----------------hashmap--------------------------------

        HashMap<Integer,String> hm=**new** HashMap<Integer,String>();

        hm.put(100,"Amit");

        hm.put(104,"Amit");

        hm.put(101,"Vijay");

        hm.put(102,"Rahul");

        System.out.println("-----------Hash map-----------");

**for** (Map.Entry m:hm.entrySet()) {

            System.out.println(m.getKey()+" "+m.getValue());

        }

    }

}

16. In Java 8, explain how Hasp Map internally works?

class Key

{

String key;

Key(String key)

{

this.key = key;

}

@Override

public int hashCode()

{

return (int)key.charAt(0);

}

@Override

public boolean equals(Object obj)

{

return key.equals((String)obj);

}

}

17. Difference between fail fast and fail-safe iterator?

1. **import** java.util.HashMap;
2. **import** java.util.Iterator;
3. **import** java.util.Map;
4. **public** **class** FailFastDemo {
5. **public** **static** **void** main(String[] args)
6. {
7. Map<String, String> empName = **new** HashMap<String, String>();
8. empName.put("Sam Hanks", "New york");
9. empName.put("Will Smith", "LA");
10. empName.put("Scarlett", "Chicago");
11. Iterator iterator = empName.keySet().iterator();
12. **while** (iterator.hasNext()) {
13. System.out.println(empName.get(iterator.next()));
14. // adding an element to Map
15. // exception will be thrown on next call
16. // of next() method.
17. empName.put("Istanbul", "Turkey");
18. }
19. }
20. }

18. Can we start the thread twice?

1. **public** **class** TestThreadTwice1 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("running...");
4. }
5. **public** **static** **void** main(String args[]){
6. TestThreadTwice1 t1=**new** TestThreadTwice1();
7. t1.start();
8. t1.start();
9. }
10. }

19. What are the different ways to create a thread in java? Which one is preferred?

**import** java.io.\*;

**class** GFG **implements** Runnable {

**public** **static** **void** main(String args[])

    {

        // create an object of Runnable target

        GFG gfg = **new** GFG();

        // pass the runnable reference to Thread

        Thread t = **new** Thread(gfg, "gfg");

        // start the thread

        t.start();

        // get the name of the thread

        System.out.println(t.getName());

    }

    @Override **public** **void** run()

    {

        System.out.println("Inside run method");

    }

}

20. What are the different states a thread will go through?

1. **class** Thread1 **extends** Thread
2. {
4. **public** **void** run()
5. {
6. System.out.println("Thread 1");
7. System.out.println("i in Thread 1 ");
8. **for** (**int** i = 1; i <= 5; i++)
9. {
10. System.out.println("i = " + i);
11. **try**
12. {
13. Thread.sleep(1000);
14. }
15. **catch** (InterruptedException e)
16. {
17. e.printStackTrace();
18. }
19. }
20. System.out.println("Thread 1 Completed.");
21. }
22. }
24. /\* Thread 2 \*/
25. **class** Thread2 **extends** Thread
26. {
27. **public** **void** run()
28. {
29. System.out.println("Thread 2");
30. System.out.println("i in Thread 2 ");
31. **for** (**int** i = 1; i <= 5; i++)
32. {
33. System.out.println("i = " + i);
34. }
35. System.out.println("Thread 2 Completed.");
36. }
37. }
39. /\* Driver code \*/
40. **public** **class** ThreadDemo
41. {
42. **public** **static** **void** main(String[] args) {
43. // life cycle of Thread
44. // Thread's New State
45. Thread1 t1 = **new** Thread1();
46. Thread2 t2 = **new** Thread2();
47. // Both the above threads are in runnable state
48. // Running state of Thread1 and Thread2
49. t1.start();
50. // Move control to another thread
51. t2.yield();
52. // Blocked State Thread1
53. **try**
54. {
55. t1.sleep(1000);
56. }
57. **catch** (InterruptedException e)
58. {
59. e.printStackTrace();
60. }
61. t2.start();
62. System.out.println("Main Thread End");
63. }
64. }

21. What is Serialization? How do we achieve it?

**import** java.io.\*;

**class** Demo **implements** java.io.Serializable

{

**public** **int** a;

**public** String b;

    // Default constructor

**public** Demo(**int** a, String b)

    {

**this**.a = a;

**this**.b = b;

    }

}

**class** Test

{

**public** **static** **void** main(String[] args)

    {

        Demo object = **new** Demo(1, "geeksforgeeks");

        String filename = "file.ser";

        // Serialization

**try**

        {

            //Saving of object in a file

            FileOutputStream file = **new** FileOutputStream(filename);

            ObjectOutputStream out = **new** ObjectOutputStream(file);

            // Method for serialization of object

            out.writeObject(object);

            out.close();

            file.close();

            System.out.println("Object has been serialized");

        }

**catch**(IOException ex)

        {

            System.out.println("IOException is caught");

        }

        Demo object1 = **null**;

        // Deserialization

**try**

        {

            // Reading the object from a file

            FileInputStream file = **new** FileInputStream(filename);

            ObjectInputStream in = **new** ObjectInputStream(file);

            // Method for deserialization of object

            object1 = (Demo)in.readObject();

            in.close();

            file.close();

            System.out.println("Object has been deserialized ");

            System.out.println("a = " + object1.a);

            System.out.println("b = " + object1.b);

        }

**catch**(IOException ex)

        {

            System.out.println("IOException is caught");

        }

**catch**(ClassNotFoundException ex)

        {

            System.out.println("ClassNotFoundException is caught");

        }

    }

}

22. What is immutable class? Is String class immutable?

**import** java.util.HashMap;

**import** java.util.Map;

// Class 1

// An immutable class

**final** **class** Student {

    // Member attributes of final class

**private** **final** String name;

**private** **final** **int** regNo;

**private** **final** Map<String, String> metadata;

    // Constructor of immutable class

    // Parameterized constructor

**public** Student(String name, **int** regNo,

                   Map<String, String> metadata)

    {

        // This keyword refers to current instance itself

**this**.name = name;

**this**.regNo = regNo;

        // Creating Map object with reference to HashMap

        // Declaring object of string type

        Map<String, String> tempMap = **new** HashMap<>();

        // Iterating using for-each loop

**for** (Map.Entry<String, String> entry :

             metadata.entrySet()) {

            tempMap.put(entry.getKey(), entry.getValue());

        }

**this**.metadata = tempMap;

    }

    // Method 1

**public** String getName() { **return** name; }

    // Method 2

**public** **int** getRegNo() { **return** regNo; }

    // Note that there should not be any setters

    // Method 3

    // User -defined type

    // To get meta data

**public** Map<String, String> getMetadata()

    {

        // Creating Map with HashMap reference

        Map<String, String> tempMap = **new** HashMap<>();

**for** (Map.Entry<String, String> entry :

**this**.metadata.entrySet()) {

            tempMap.put(entry.getKey(), entry.getValue());

        }

**return** tempMap;

    }

}

// Class 2

// Main class

**class** GFG {

    // Main driver method

**public** **static** **void** main(String[] args)

    {

        // Creating Map object with reference to HashMap

        Map<String, String> map = **new** HashMap<>();

        // Adding elements to Map object

        // using put() method

        map.put("1", "first");

        map.put("2", "second");

        Student s = **new** Student("ABC", 101, map);

        // Calling the above methods 1,2,3 of class1

        // inside main() method in class2 and

        // executing the print statement over them

        System.out.println(s.getName());

        System.out.println(s.getRegNo());

        System.out.println(s.getMetadata());

        // Uncommenting below line causes error

        // s.regNo = 102;

        map.put("3", "third");

        // Remains unchanged due to deep copy in constructor

        System.out.println(s.getMetadata());

        s.getMetadata().put("4", "fourth");

        // Remains unchanged due to deep copy in getter

        System.out.println(s.getMetadata());

    }

}

23. Do immutable classes thread safe? If yes then how?

public class ImmutableValue{

private int value = 0;

public ImmutableValue(int value){

this.value = value;

}

public int getValue(){

return this.value;

}

}

24. Can we call the garbage collector explicitly? Will it trigger the garbage collector?

**class** Employee {

**private** **int** ID;

**private** String name;

**private** **int** age;

**private** **static** **int** nextId = 1;

    // it is made static because it

    // is keep common among all and

    // shared by all objects

**public** Employee(String name, **int** age)

    {

**this**.name = name;

**this**.age = age;

**this**.ID = nextId++;

    }

**public** **void** show()

    {

        System.out.println("Id=" + ID + "\nName=" + name

                           + "\nAge=" + age);

    }

**public** **void** showNextId()

    {

        System.out.println("Next employee id will be="

                           + nextId);

    }

}

**class** UseEmployee {

**public** **static** **void** main(String[] args)

    {

        Employee E = **new** Employee("GFG1", 56);

        Employee F = **new** Employee("GFG2", 45);

        Employee G = **new** Employee("GFG3", 25);

        E.show();

        F.show();

        G.show();

        E.showNextId();

        F.showNextId();

        G.showNextId();

        { // It is sub block to keep

            // all those interns.

            Employee X = **new** Employee("GFG4", 23);

            Employee Y = **new** Employee("GFG5", 21);

            X.show();

            Y.show();

            X.showNextId();

            Y.showNextId();

        }

        // After countering this brace, X and Y

        // will be removed.Therefore,

        // now it should show nextId as 4.

          // Output of this line

        E.showNextId();

        // should be 4 but it will give 6 as output.

    }

}

25. What are Java 8 features? Explain all of them with examples?

importjava.util.ArrayList;

importjava.util.List;

**public** **class** Main {

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

        List<String> subList = **new** ArrayList<String>();

        subList.add("Maths");

        subList.add("English");

        subList.add("French");

        subList.add("Sanskrit");

        subList.add("Abacus");

        System.out.println("------------Subject List--------------");

        subList.forEach(sub -> System.out.println(sub));

  }

}  
26. How to make a pure singleton?

**class** Singleton {

    // Static variable reference of single\_instance

    // of type Singleton

**private** **static** Singleton single\_instance = **null**;

    // Declaring a variable of type String

**public** String s;

    // Constructor

    // Here we will be creating private constructor

    // restricted to this class itself

**private** Singleton()

    {

        s = "Hello I am a string part of Singleton class";

    }

    // Static method

    // Static method to create instance of Singleton class

**public** **static** Singleton getInstance()

    {

**if** (single\_instance == **null**)

            single\_instance = **new** Singleton();

**return** single\_instance;

    }

}

// Class 2

// Main class

**class** GFG {

    // Main driver method

**public** **static** **void** main(String args[])

    {

        // Instantiating Singleton class with variable x

        Singleton x = Singleton.getInstance();

        // Instantiating Singleton class with variable y

        Singleton y = Singleton.getInstance();

        // Instantiating Singleton class with variable z

        Singleton z = Singleton.getInstance();

        // Printing the hash code for above variable as

        // declared

        System.out.println("Hashcode of x is "

                           + x.hashCode());

        System.out.println("Hashcode of y is "

                           + y.hashCode());

        System.out.println("Hashcode of z is "

                           + z.hashCode());

        // Condition check

**if** (x == y && y == z) {

            // Print statement

            System.out.println(

                "Three objects point to the same memory location on the heap i.e, to the same object");

        }

**else** {

            // Print statement

            System.out.println(

                "Three objects DO NOT point to the same memory location on the heap");

        }

    }

}

27. How to make a singleton synchronized?

**public** **class** GFG

{

  // public instance initialized when loading the class

**private** **static** **final** GFG instance = **new** GFG();

**private** GFG()

  {

    // private constructor

  }

**public** **static** GFG getInstance(){

**return** instance;

    }

}