1. What are generic methods?

a) Generic methods are methods that take void parameters

b) Generic methods are the methods defined in a generic class

c) Generic methods are the methods that extend generic class methods

**d) Generic methods are methods that introduce their own type parameters**

2) Which of these type parameters is used for a generic class to return and accept a number?

a) K

**b) N**

c) T

d) V

3) Which of these type of parameters is used for a generic class to return and accept any type of object?

A) K

b) N

**c) T**

d) V

4) Which of the following types of reference cannot be generic?

a) Interface

b) Inner classes

**c) Inner class anonymous**

d) All the answers are true

5)Which of these types cannot be used to initiate a generic type?

a) Float class

b) Integer class

c) Collections

**d) Primitive Types**

6) Which of these instance cannot be created?

a) Collection instances

b) Integer instance

**c) Generic type instance**

d) Generic class instance

7) What is meant by the term generics?

a) class

b) structure

c) interface

**d) parameterized types**

8) Which of the following allows us to call generic methods as a normal method?

a) Interface

b) Inner class

**c) Type Interface**

d) All of the mentioned

9) Which of the following is incorrect statement regarding the use of generics and parameterized types in Java?

a) Generics provide type safety by shifting more type checking responsibilities to the compiler

b) Generics and parameterized types eliminate the need for down casts when using Java Collections

**c) When designing your own collections class (say, a linked list), generics and parameterized types allow you to achieve type safety with just a single class definition as opposed to defining multiple classes**

d) All of the mentioned

10) What will be the output of the following Java program?

import java.util.\*;

public class genericstack

{

Stack stk = new Stack ();

public void push(E obj)

{

stk.push(obj);

}

public E pop()

{

E obj = stk.pop();

return obj;

}

}

class Output

{

public static void main(String args[])

{

genericstack gs<String> = new genericstack<String> ();

gs.push("Hello");

System.out.println(gs.pop());

}

}

a) H

**b) Hello**

c) Runtime Error

d) Compilation Error

11) Why are generics used?

a) Generics make code more fast

b) Generics make code more optimised and readable

**c) Generics add stability to your code by making more of your bugs detectable at compile time**

d) Generics add stability to your code by making more of your bugs detectable at a runtime

12) What will be the output of the following Java program?

import java.util.\*;

public class genericstack

{

Stack stk = new Stack ();

public void push(E obj)

{

stk.push(obj);

}

public E pop()

{

E obj = stk.pop();

return obj;

}

}

class Output

{

public static void main(String args[])

{

genericstack gs = new genericstack();

gs.push("Hello");

System.out.print(gs.pop() + " ");

genericstack gs = new genericstack();

gs.push(36);

System.out.println(gs.pop());

}

}

a) Error

b) Hello

c) 36

**d) Hello 36**

13) Which of these Exception handlers cannot be type parameterized?

a) Catch

b) Throw

c) Throws

**d) All of the mentioned**

14) Generics does not work with?

a) Set

b) List

c) Tree

**d) Array**

15) Which of these instances cannot be created?

a) Integer Instance

b) Generic Class Instance

**c) Generic Type Instance**

d) Collection Instances

16) What will be the output of the following program?

public class UseGenerics {

public static void main(String args[]){

MyGen<Integer> m = new MyGen<Integer>();

m.set("merit");

System.out.println(m.get());

}

}

class MyGen<T>

{

T var;

void set(T var)

{

this.var = var;

}

T get()

{

return var;

}

}

a) merit

**b) 0**

c) Compilation Error

d) Runtime Error

17) Which version of Java introduced annotation?

**a) Java 5**

b) Java 6

c) Java 7

d) Java 8

18) Annotation type definition looks similar to which of the following?

a) Method

b) Class

**c) Interface**

d) Field

19) Which of the following is not pre defined annotation in Java?

a) @Deprecated

**b) @Overriden**

c) @SafeVarags

d) @FunctionInterface

20) Using which annotation non visible or private method can be tested?

**a) @VisibleForTesting**

b) @NonVisibleForTesting

c) @Visible

d) @NonVisible

21) Which is the Parent class of annotation class?

a) Class

**b) Object**

c) Main

d) Super

22) At what line will there be a compilation error?

public class Generics {

public static void main(String[] args) {

B b = new B(); /\* LINE A \*/

C c = b.process(new C()); /\* LINE B \*/

B<C> b2 = new B<C>(); /\* LINE C \*/

C c2 = b2.process(new C()); /\* LINE D \*/

}

}

interface A {

int count();

void show();

}

class B<T extends A> {

T process(T t) {

t.count();

t.show();

return t;

}

}

class C implements A {

public int count() { return 25; }

public void show() { System.out.print("Class C"); }

}

a) LINE A

**b) LINE B**

c) LINE C

d) LINE D

23) What will be the output of the following program?

import java.util.\*;

public class SampleDemo {

public static void main(String args[]) {

Sample <Integer> obj = new Sample<Integer>();

obj.push(36);

System.out.println(obj.pop());

}

}

class Sample<E>

{

Stack <E> ob = new Stack <E>();

public void push(E obj) {

ob.push(obj);

}

public E pop() {

E obj = ob.pop();

return obj;

}

}

**a) 36**

b) Some Other Output

c) Compilation Error

d) Runtime Error

24) What will be the output of the following program?

public class GenericsWithObjectsDemo {

public static void main(String args[])

{

GenericsWithObjects<double> doubleObject = new GenericsWithObjects<double>(12.0);

doubleObject.print();

}

}

class GenericsWithObjects<T>

{

T obj;

GenericsWithObjects(T obj)

{

this.obj = obj;

}

void print()

{

System.out.println(obj);

}

}

a) 12.0

b) Some other output

**c) Compilation Error**

d) Runtime Error

25) What will be the output of the following program?

public class GenericsWithObjectsDemo {

public static void main(String args[]) {

GenericsWithObjects<Double> doubleObject = new GenericsWithObjects<Double>(12.0);

doubleObject.print();

GenericsWithObjects<String> stringObject = new GenericsWithObjects<String>("MC");

stringObject.print();

}

}

class GenericsWithObjects<T> {

Object obj;

GenericsWithObjects(Object obj) {

this.obj = obj;

}

void print() {

System.out.println(obj);

}

}

**a)12.0**

**MC**

b)12.0

12.0

c) Compilation Error

d) Runtime Error

26) Which of the following is an invalid Java generics syntax?

a) ArrayList<String> al1 = new ArrayList<String>();

b) List<String> al2 = new ArrayList<String>();

c) Collection<String> al3 = new ArrayList<String>();

**d) ArrayList<Object> al4 = new ArrayList<String>();**

27) For the below Test class which of the following way of object creation is invalid?

class Test<T extends Number> { }

a) Test<Integer> t1 = new Test<Integer>();

**b) Test<String> t2 = new Test<String>();**

c) Test<Double> t3 = new Test<Double>();

d) Test<Number> t4 = new Test<Number>();

28) Which of the following is/are valid declarations?

**a) class Test<T extends Number & Runnable> { }**

b) class Test<T extends Runnable & Number> { }

c) class Test<T extends Number & Thread> { }

d) All of these

29) For the below Test class which of the following way of object creation is invalid?

class Test<T extends Runnable> { }

class MyThread1 extends Thread { }

class MyThread2 implements Runnable {

public void run(){}

}

a) Test<Runnable> t1 = new Test<Runnable>();

b) Test<MyThread1> t2 = new Test<MyThread1>();

c) Test<MyThread2> t3 = new Test<MyThread2>();

**d) None of these**

30) Which character is used to represent any type or wildcard element?

a) >

**b) ?**

c) ,

d) |