JAVA IA 2 Q&A

Q. 1. What is a constructors? Explain its types.

- In Java, a constructor is a block of codes similar to the method.
- It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.
- There are three types of constructors in Java:

  no-arguement, parameterized and Default constructor

1. No-argument constructor

- A constructor that has no parameter is known as the No-argument or Zero argument constructor.
- If we don't define a constructor in a class, then the compiler creates a constructor(with no arguments) for the class.
- And if we write a constructor with arguments or no arguments then the compiler does not create a default constructor.

2. Parameterized Constructor

- A constructor that has parameters is known as parameterized constructor.
- If we want to initialize fields of the class with our own values, then use a parameterized constructor.

3. Default Constructor

- A constructor that has no parameters is known as default the

constructor. A default constructor is invisible.

- And if we write a constructor with arguments or no arguments then the compiler does not create a default constructor
- It is taken out. It is being overloaded and called a parameterized constructor.
- The default constructor changed into the parameterized constructor.
- But Parameterized constructor can't change the default constructor.

Q. 2. Write a program to demonstrate the concept of constructors.

```
class Main {
  private String name;

  // constructor
  Main() {
    System.out.println("Constructor Called:");
    name = "Programiz";
  }

  public static void main(String[] args) {

    // constructor is invoked while
    // creating an object of the Main class
    Main obj = new Main();
    System.out.println("The name is " + obj.name);
  }
}
```

Run Code »

Output:

```
Constructor Called:
The name is Programiz
```

Q. 3. Write short note on Arrays with a suitable example.

- An array is defined as a sequence of objects of the same data type.
- All the elements of an array are either of type int (whole numbers), or all of them are of type char, or all of them are of floating decimal point type, etc.
- An array cannot have a mixture of different data types as its elements.
- Also, array elements cannot be functions; however, they may be pointers to functions. In computer memory, array elements are stored in a sequence of adjacent memory blocks.
- Since all the elements of an array are of same data type, the memory blocks allocated to elements of an array are also of same size. Each element of an array occupies one block of memory.
- The size of memory blocks allocated depends on the data type and it is same as for different data types.
- The declaration of array includes the type of array that is the type of value we are going to store in it, the array
  name and maximum number of elements.
- Datatype array_name [dimensions] = {element1,element2,...,element}
- Examples:

```
1 short val[200];
2 val[12] = 5;
```

Q. 4. Distinguish between method overloading and method overriding.

Method Overloading	Method Overriding
Method overloading is a compile-time polymorphism.	Method overriding is a run-time polymorphism.
It helps to increase the readability of the program.	It is used to grant the specific implementation of the method which is already provided by its parent class or superclass.
It occurs within the class.	It is performed in two classes with inheritance relationships.
Method overloading may or may not require inheritance.	Method overriding always needs inheritance.
In method overloading, methods must have the same name and different signatures.	In method overriding, methods must have the same name and same signature.
In method overloading, the return type can or can not be the same, but we just have to change the parameter.	In method overriding, the return type must be the same or co-variant.
Static binding is being used for overloaded methods.	Dynamic binding is being used for overriding methods.
Poor Performance due to compile time polymorphism.	It gives better performance. The reason behind this is that the binding of overridden methods is being done at runtime.
Private and final methods can be overloaded.	Private and final methods can't be overridden.
Argument list should be different while doing method overloading.	Argument list should be same in method overriding.

Q. 5. What is compile time polymorphism? Explain.

- Compile-time polymorphism is also known as static polymorphism or early binding.
- Compile-time polymorphism is a polymorphism that is resolved during the compilation process.
- Compile-time polymorphism is Method Overloading.
- Overloading of methods is called through the reference variable of a class.
- Compile-time polymorphism is achieved by method overloading ~~and operator overloading~~.
- All the methods of compile-time polymorphism get called or invoked during the compile time.
- Compile-time polymorphism occurs within the class.
- Compile-time polymorphism may or may not require inheritance.
- Static binding is being used for Compile-time polymorphism.
- The following are the key advantage of compile-time polymorphism:

i. Debugging the code becomes easier
ii. It helps us to reuse the code.
iii. Binding of the code is performed during compilation time
iv. Overloading of the method makes the working of the code simpler and more efficient.

Q. 9. What is run time polymorphism? Explain.

- Polymorphism is a technique wherein a single action can be performed in two different ways. The term polymorphism is derived from two Greek words, "poly" and "morphs" which mean many and

forms, respectively.

- In real-time, polymorphism can be explained as the different roles played by a single person.
- For instance, a man can be a son, husband, father, etc.
- In other words, a person has many features and implements each one of them according to the situation.
- Polymorphism is one of the key aspects of Object-Oriented Programming (OOP).
- Run-time polymorphism is method overriding.
- In runtime polymorphism, the compiler resolves the object at run time and then it decides which function call should be associated with that object.
- It is also known as dynamic or late binding polymorphism.
- This type of polymorphism is executed through virtual functions and function overriding.
- All the methods of runtime polymorphism get invoked during the run time.
- Runtime polymorphism is known to be better for dealing with complex problems since all the methods and details turn up during the runtime itself.

Q. 6. Write a program to demonstrate the user defined packages.

Code:-

```
// Java program to create a user-defined
// package and function to print
// a message for the users
package example;
```

```
// Class
public class gfg
{
	public void show()
	{
		System.out.println("Hello geeks!! How are you?");
	}
	public static void main(String args[])
	{
		gfg obj = new gfg();
		obj.show();
	}
}
```

Output:-

```
Hello geeks!! How are you?
```

Q. 7. Write a program to demonstrate the user defined exception handling.

Code:-

```
import java.util.Scanner;
class NegativeAmtException extends Exception
{
	String msg;
	NegativeAmtException(String msg)
	{
		this.msg=msg;
	}
	public String toString()
	{
		return msg;
	}
}
public class User_Defined_Exception
```

```
{
	public static void main(String[] args)
	{
		Scanner s=new Scanner(System.in);
		System.out.print("Enter Amount:");
		int a=s.nextInt();
		try
		{
			if(a<0)
			{
				throw new NegativeAmtException("Invalid
Amount");
			}
			System.out.println("Amount Deposited");
		}
		catch(NegativeAmtException e)
		{
			System.out.println(e);
		}
	}
}
```

Output:-

```
$ javac User_Defined_Exception.java
$ java User_Defined_Exception


Enter Amount:-5
Invalid Amount
```

Q. 8. Explain the features of Java.

1. Inspired by C and C++

   Java is inspired by C and C++. The syntax of Java is similar to these languages but the languages are quite different. Java inherits many features from C and C++. Compared to C++, Java code runs a bit

slower but it is more portable and offers better security features.

2. Simple and Familiar
   Java programming language is simple to learn, understand, read, and write. Java programs are easy to create and implement compared to other programming languages such as C and C++. If you are familiar with the basic principles of programming or the concept of OOP (object-oriented programming), it would be easy to master Java.

3. Object-Oriented
   Java is a fully object-oriented language, unlike C++ which is semi object-oriented. It supports every OOP concept such as Abstraction, Encapsulation, Inheritance, Polymorphism. Java programs are developed using classes and objects.

4. Platform Independent
   Java 's platform independence means that Java programs compiled on one machine or operating system can be executed on any other machine or operating system without modifications. Java supports WORA (Write Once, Run Anywhere), which means that programmers can develop applications in one operating system and run on any other without any modification.

5. Dynamic
   Java is more dynamic compared to C and C++. It can adapt to its evolving environment. It allows programmers to dynamically link new class libraries, objects, and methods. Java programs can have a large amount of
   run-time information that can be used to resolve accesses to objects.

6. Robust
   Java is a robust language that can handle run-time errors as it checks the code during the compile and runtime. If any runtime error is identified by the JVM, it will not be passed directly to the underlying system. Instead, it will immediately terminate the program and stop it

from causing any harm to the system.

7. Secure
   Java is a secure language that ensures that programs cannot gain access to memory locations without authorization. It has access modifiers to check memory access. Java also ensures that no viruses enter an applet.

8. High Performance
   Java offers high performance as it used the JIT (Just In Time) compiler. The compiler only compiles that method which is being called. The JIT enhances the performance of interpreting byte code by caching interpretations.

9. Portable
   Due to the concept of Write Once Run Anywhere (WORA) and platform independence, Java is a portable language. By writing once through Java, developers can get the same result on every machine.

10. Automatic Garbage Collection
    The most common and critical problems in C/C++ were memory leaks.FYI, memory leaks are a problem wherein a piece of memory that is no longer in use, fails to be freed.

Q.10.Explain any 5 types of combination of try, catch and finally from exception handling.

TYPE - 1

```
try
{
}
catch(X e)
{
```

}

Compiles FINE

Type - 6

```
try
{
}
```

Compile Time Error : try without catch or finally

Type - 7

```
catch(X e)
{
}
```

Compile Time Error : catch without try

Type - 8

```
finally
{
}
```

Compile Time Error : finally without try

Type - 9

```
try
{
}
finally
{
}
```

Compile FINE