**Interface**

**Assignment**

1. **What is an interface in Java?**

**Ans:** Interface is a Java Feature, it will allow only abstract methods. In Java applications, for interfaces, we are able to create only reference variables, we are unable to create objects.In the case of interfaces, by default, all the variables are "public static final".In the case of interfaces, by default, all the methods are "public and abstract".In Java applications, constructors are possible in classes and abstract classes but constructors are not possible in interfaces.Interfaces will provide more shareability in Java applications when compared with classes and abstract classes.

Syntax:

interface <interface\_name>

{

     // declare constant fields

     // declare methods that abstract

     // by default.

}

1. **Which modifiers are allowed for methods in an Interface?Explain with an example?**

**Ans:** Only public and abstract modifiers are allowed for methods in an interfaces.

Example:

**package** Interfaces;

**abstract** **class** Vehicle

{

**abstract** **public** **int** getNumberOfWheel();

}

**class** Truck **extends** Vehicle

{

**public** **int** getNumberOfWheel() { **return** 12; }

}

**class** Auto **extends** Vehicle

{

**public** **int** getNumberOfWheel() { **return** 3; }

}

**public** **class** AbsModExam

{

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

{

Truck b = **new** Truck();

Auto a = **new** Auto();

System.***out***.println("Number of wheels in Truck is"+ " " + b.getNumberOfWheel());

System.***out***.println("Number of wheels in Auto is"+ " " + a.getNumberOfWheel());

}

1. **What is the use of interface in Java?Or, why do we use an interface in Java?**

**Ans:** To define a protocol of behavior that can be implemented by any class anywhere in the class hierarchy.

**There are mainly three reasons to use interface. They are given below:-**

1. It is used to achieve abstraction.
2. By interface, we can support the functionality of multiple inheritance.
3. It can be used to achieve loose coupling.
4. **What is the difference between abstract class and interface in Java?**

**Ans:**

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1. Abstract class can have abstract and non-abstract methods. | 1. Interface can have only abstract methods. Since Java 8, it can have default and static methods also. |
| 1. Abstract class doesn't support multiple inheritance. | 1. Interface supports multiple inheritance. |
| 1. Abstract class can have final, non-final, static and non-static variables. | 1. Interface has only static and final variables. |
| 1. Abstract class can provide the implementation of interface. | 1. Interface can't provide the implementation of abstract class. |
| 1. The abstract keyword is used to declare abstract class. | 1. The interface keyword is used to declare interface. |
| 1. An abstract class can extend another Java class and implement multiple Java interfaces. | 1. An interface can extend another Java interface only. |
| 1. An abstract class can be extended using keyword "extends". | 1. An interface can be implemented using keyword "implements". |
| 1. A Java abstract class can have class members like private, protected, etc. | 1. Members of a Java interface are public by default. |
| 1. Example: public abstract class Shape   {  public abstract void draw(); } | 1. Example: public interface Drawable   {  void draw(); } |

**Lambda Expression**

**Assignment**

1. **What is the lambda expression of Java ?**

**Ans:** Lambda Expression is just an anonymous(nameless) function. That means the function which doesn’t have the name, return type and access modifiers. Lambda Expression also known as anonymous functions or closures.

(parameter-list) -> {body}

Java lambda expression is consisted of three components.

1. **Argument-list:**  It can be empty or non-empty as well.
2. **Arrow-token:**  It is used to link arguments-list and body of expression.
3. **Body:** It contains expressions and statements for lambda expression.

**Syntax:-** Three Syntax

* 1. **No Parameter Syntax**

() -> { //Body of no parameter lambda }

* 1. **One Parameter Syntax**

p1 -> { //Body of single parameter lambda }

* 1. **Two Parameter Syntax**

(p1,p2) -> { //Body of multiple parameter lambda }

Example:-

public void m1()

{

System.out.println(“hello”);

}

Equivalent lambda expressions:-

() ->System.out.println(“hello”);

1. Can you pass lambda expressions to a method? When?

Ans: Yes, you can pass a lambda expression to a method provided it is expecting a functional interface. For example, if a method is accepting a Runnable, Comparable or Comparator then you can pass a lambda expression to it because all these are functional interfaces in Java 8.

1. **What is the functional interface in Java?**

**Ans:** If an interface contains only one abstract method, such types of interfaces are called functional interfaces and the method is called functional method or single abstract method(SAM).

1. Runnable -It contains only run() method
2. Comparable -It contains only compareTo() method
3. ActionListener -It contains only actionPerformed()
4. Callable- It contains only call()method

In Java 8 ,SunMicroSystem introduced @FunctionalInterface annotation to specify that the interface is

FunctionalInterface.

Ex:

@FunctionalInterface

Interface Interf

{

public void m1();

}

1. **Why do we use lambda expressions in Java?**

**Ans:** We use lambda expressions in Java because:-

* + 1. To provide the implementation of Functional interface.
    2. Less coding.

1. **Is it mandatory for a lambda expression to have parameters?**

**Ans:** No, it's not mandatory for a lambda expression to have parameters, you can define a lambda expression without parameters as shown below:

() -> System.out.println("lambdas without parameter");

You can pass this code to any method which accepts a functional interface.

**Exception Handling**

**Assignment**

1. **Explain different types of Errors in Java?**

**Ans:** In any programming language we categorise errors into 2 type:-

1. Syntax Error/CompileTime Mistake
2. Logical Error/RunTimeMistakes
3. Syntax error/CompileTime Mistake-:It refers to the mistakes done by the programmer with respect to syntax. These mistakes are identified by the compiler, so we say it as“CompileTimeMistake”.
4. Logical Error/RunTimeMistake:-It refers to the mistakes done by the programmer in terms of writing a logic. These mistakes are identified by jvm during the execution of a program, so we say it as “RunTimeMistake”.
5. **What is an Exception in Java?**

**Ans:** An unwanted/expected event that disturbs the normal flow of execution of a program is called "Exception handling". The main objective of Exception handling is to handle the exception. It is available for graceful termination of program.

1. **How can you handle exceptions in Java?Explain with an example?**

**Ans:** Exception handling can be performed using:

Try: the set of statements or code which requires monitoring for an exception is kept under this block.

Catch: this block catches all exceptions that were trapped in the try block.

Finally: this block is always performed irrespective of the catching of exceptions in the try or catch block.

Ex: class Launch

     {

         public static void main(String args[])

         {

             try

             {

                 System.out.print("Hello" + " " + 1 / 0);

             }

             catch(ArithmeticException e)

            {

         System.out.print("World");

             }

         }

    }

1. **Why do we need exception handling in Java?**

**Ans**: If there is no try and catch block while an exception occurs, the program will terminate. Exception handling ensures the smooth running of a program without program termination**.**

1. **What is the difference between exception and error in Java?**

**Ans:** Difference between Errors and Exceptions in Java

|  |  |
| --- | --- |
| **Errors** | **Exceptions** |
| 1. The error indicates trouble that primarily occurs due to the scarcity of system resources. | 1. The exceptions are the issues that can appear at runtime and compile time. |
| 1. It is not possible to recover from an error. | 1. It is possible to recover from an exception. |
| 1. In java, all the errors are unchecked. | 1. In java, the exceptions can be both checked and unchecked. |
| 1. The system in which the program is running is responsible for errors. | 1. The code of the program is accountable for exceptions. |
| 1. They are described in the java.lang.Error package. | 1. They are described in java.lang.Exception package |

1. **Name the different types of exceptions in Java?**

**Ans:** There are two kinds of exceptions in Java:

1. **Checked exceptions**: These are the exceptions that are checked by the compiler at compile time. If a method throws a checked exception, then the caller of the method must either handle the exception or declare it in the throws clause.
2. **Unchecked exceptions**: These are the exceptions that are not checked by the compiler at compile time. They include runtime exceptions and errors.
3. **Can we just use try instead of finally and catch blocks?**

**Ans:** No, doing so will show a compilation error. Catch or finally block must always accompany try block. We can remove either finally block or catch block, but never both.