**Task-11 : Java Exception and Error Handling**

1. **What are the four access modifiers available in java and what is their significance in terms of class, method, and variable accessibility?**

***Answer:***

  Access Specified helps to restrict the scope of a class, constructor, variable, method, or data member.

Types of Access Modifier in Java

1. Default - No keyword required
2. Private
3. Protected
4. Public

**Default Access Modifier:**

      When no access modifier is specified for a class, method or data member-It is said to be having the default access modifier by default. The data member, classes, or methods that are not declared using any access modifiers .Therefore having default access modifier are accessible only within the same package.

**Private Access Modifier:**

     The private access modifier is specified using the keyword private. The methods or data members declared as private are accessible only within the class in which they are declared.

**Protected Access Modifier:**

    The protected access modifier is specified using the keyword protected. The methods or data members declared as protected are accessible within the same package or subclasses in different packages.

**Public Access Modifier:**

           The public access modifier has the widest scope among all other access modifiers. Classes, methods, or data members that are declared as public are accessible from everywhere in the program. There is no restriction on the scope of public data members.

**Significance:**

In java, you can access specifies to protect both a classes variables and its methods when you declare them. The reason becomes more apparent when you have to maintain a larger project.

When a method or variable is public, you have to be careful when you make changes to it, because you never know which parts of the codebase rely on its exact behaviour.

But when a variable or method is private, you know that it is not used outside of the class. That means there is a lot less code you have to pay attention to when you make changes.

By making class features private and public, you clearly separate the interface to the outside world from the internals. The less you exposes to the outside world, the more freedom you have with what the internal implementation does.

When you, for example, always make variables private and accessed them through getters and setters, you can later change them from a variable to a computed value, and then even later add caching to the computation for performance reasons. When it would be a public variable, you would have to change code everywhere the variable is used. But when you expose it to the outside world through getters and setters, all other code can keep using the class as if nothing had changed.

Here is the basic algorithm for using access modifiers in java:

* Define a class: Create a class that represents the object you want to manage.
* Define instances variables: Within the class, define instance variables that represent the data you want to manage.
* Specify an access modifier: For each instance variable, specify an access modifier that determines the visibility of the variable. The three main access modifiers in Java are private, protected, and public.
* Use private for variables that should only be accessible within the class: If you want to prevent access to a variable from outside the class, use the private access modifier. This is the most restrictive access modifier and provides the greatest level of encapsulation.
* Use protected for variables that should be accessible within the class and its subclasses: If you want to allow access to a variable from within the class and its subclasses, use the protected access modifier. This is less restrictive than private and provides some level of inheritance.
* Use public for variables that should be accessible from anywhere: If you want to allow access to a variable from anywhere, use the public access modifier. This is the least restrictive access modifier and provides the least amount of encapsulation.
* Use accessor and mutator methods to manage access to the variables:In order to access and modify the variables, use accessor (getter) and mutator (setter) methods, even if the variables have a public access modifier. This provides a level of abstraction and makes your code more maintainable and testable.

In short, the access modifiers can be tabulated as:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Modifier | Within Class | Within package | Outside package by subclass only | Outside package |
| Private | Y | N | N | N |
| Default | Y | Y | N | N |
| Protected | Y | Y | Y | N |
| Public | Y | Y | Y | Y |

**2.  What is the difference between Exception and Error?**

***Answer:***

The differences are listed below:

**Exception:**

* Exception can be recovered by using the try-catch block.
* It can be classified into two categories i.e. checked and unchecked.
* It occurs at compile time or run time.
* It belongs to java.lang.Exception package.
* Only checked exceptions are known to the complier.
* It is mainly caused by the application itself.
* Checked Exception: SQL Exception, IO Exception
* Unchecked Exception : ArrayIndexOutOfBoundException,
* Eg: NullPOinterException, ArithmaticException etc.

**Error:**

* Error cannot be recovered.
* All error in java unchecked.
* It occurs at run time.
* It belongs to java.lang.Error package.
* Error will not be known the complier.
* It is mostly caused by environment in which application is running.
* Eg: Java.lang.StackOverFlow, Java.lang.OutOfMemoryError etc.

**Tabulation of Difference between Error and Exception in Java**

|  |  |
| --- | --- |
| **Error** | **Exception** |
| An error cannot be handled at runtime. | An exception can be handled at runtime |
| An error can occur both at compile time and during runtime. | Although all the Exceptions occur at runtime. But checked Exceptions can be detected at compile time. |
| There are 3 types of Errors: **Syntax Error**, **Runtime Error** and **Logical Error** | There are 2 types of Exceptions: **Checked Exceptions** and **Unchecked Exceptions** |
| An error has the capacity to terminate your program and maybe your system as well. | An exception has the capacity to distract the normal flow of the program and change its direction to somewhere else when an exceptional case has occurred. |
| An Error is such an event that no one can control or guess when it is going to happen. | An Exception can be guessed, handled, and utilized in order to change the original flow of the program. |
| An Error can be thought of as an explosion that happens when there is no defence or checks against a particular failure condition. | An Exception can be thought of as a last line of defence to prevent errors. |
| There are 3 types of Errors: Syntax Error, Runtime Error and Logical Error | There are 2 types of Exceptions: Checked Exceptions and Unchecked Exceptions |
| An error has the capacity to terminate your program and maybe your system as well. | An exception has the capacity to distract the normal flow of the program and change its direction to somewhere else when an exceptional case has occurred. |
| An Error is such an event that no one can control or guess when it is going to happen. | An Exception can be guessed, handled, and utilized in order to change the original flow of the program. |
| An Error can be thought of as an explosion that happens when there is no defence or checks against a particular failure condition. | An Exception can be thought of as a last line of defence to prevent errors. |

**3. What is the difference between checked Exception and unchecked Exception?**

***Answer:***

The differences are listed below:

**Checked Exception:**

* Exception that are checked and handled at compile time are checked exception.
* They are direct subclasses of Exception but do not inherit from runtime Exception.
* The program gives a compilation error if a method throws a checked exception and the compiler is not able to handle the exception on its own.
* A checked Exception occurs when the chances of failure are too high.
* Common checked exceptions include IOException, DataAccessException, IllegalAccessException, InterruptedException, etc.

**Unchecked Exception:**

* Exception that are not checked and handled at compile time are unchecked exceptions.
* They are a direct subclass of runtime Exception.
* The program compiles fine because the exceptions escape the notice of compiler. Exceptions occur due to errors in programming logic.
* Unchecked Exception occurs mostly due to programming mistakes.
* Common unchecked exceptions include ArithmeticExceptions, InvalidClassException, NullpointerException, etc.

**Tabulation of Difference between Checked and Unchecked Exception**

|  |  |
| --- | --- |
| **Checked Exceptions** | **Unchecked Exceptions** |
| Checked exceptions occur at compile time. | Unchecked exceptions occur at runtime. |
| The compiler checks a checked exception. | The compiler does not check these types of exceptions. |
| These types of exceptions can be handled at the time of compilation. | These types of exceptions cannot be a catch or handle at the time of compilation, because they get generated by the mistakes in the program. |
| They are the sub-class of the exception class. | They are runtime exceptions and hence are not a part of the Exception class. |
| For checked exceptions, the JVM needs the exception to catch and handle. | For Unchecked exceptions, the JVM does not require the exception to catch and handle. |
| Examples of Checked exceptions:   * File Not Found Exception * No Such Field Exception * Interrupted Exception * No Such Method Exception * Class Not Found Exception | Examples of Unchecked Exceptions:   * No Such Element Exception * Undeclared Throwable Exception * Empty Stack Exception * Arithmetic Exception * Null Pointer Exception * Array Index Out of Bounds Exception * Security Exception |

-----