Java

What is Exception

An Unwanted unexpected event that disturbs the normal flow of program is called Exception.

What is the purpose of Exception handling

To Graceful termination of the program.

What is meaning of Exception handling

Defining the alternative way to continue rest of the program normally.

Three way we handle Exception handling

1. try & catch
2. throws
3. throw keyword

Checked Exceptions In Java :

Checked exceptions are the exceptions which are checked during compilation itself. They are also called compile time exceptions. Compiler is aware of these exceptions and immediately throws the error wherever it sees the statements which may throw checked exceptions.

All sub classes of java.lang.Exception (except sub classes of RunTimeException) are checked exceptions. For example, FileNotFoundException, IOException, SQLException, ClassNotFoundException etc…

These exceptions must be handled either using try-catch blocks or using throws clause. If not handled properly, they will give compile time error.

For example,

Below code throws ClassNotFoundException which is a checked exception. But it is not handled, so it gives compile time error.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | **public** **class** CheckedException  {  **public** **static** **void** main(String[] args)      {          Class.forName("AnyClassName");            //Compile time error because          //above statement throws ClassNotFoundException which is a checked exception          //this statement must be enclosed within try-catch block or declare main method with throws clause      }  } |

Below are the correct ways to write the above code.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **public** **class** CheckedException  {  **public** **static** **void** main(String[] args)      {  **try**          {              Class.forName("AnyClassName");          }  **catch** (ClassNotFoundException ex)          {              System.out.println("ClassNotFoundException will be caught here");          }      }  } |

**OR**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **public** **class** CheckedException  {  **public** **static** **void** main(String[] args) **throws** ClassNotFoundException      {          Class.forName("AnyClassName"); |

What is the purpose of Throw keyword

To hand over our created Exception object to the JVM manually.

To throw key word we can create our own Exception

Throw Keyword

1 throw is used within the method and block.

2 In case of throw keyword we can throw only one single exception.

3 throw keyword is mainly used for runtime exception or unchecked exception.

Throws Keyword

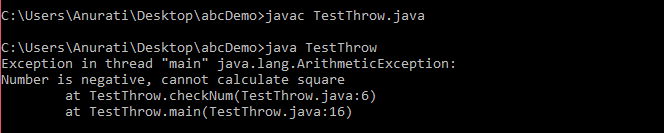
1. throws is used with the method signature.
2. throws keyword is mainly used for compile time exception or checked exception.
3. In case of throw keyword we can declare multiple exception.
4. Throws keyword is used to declare the exception i.e. it indicates the caller method that given type of exception can occur so you can handle it while calling.

Java throw Example

**TestThrow.java**

1. **public** **class** TestThrow {
2. //defining a method
3. **public** **static** **void** checkNum(**int** num) {
4. **if** (num < 1) {
5. **throw** **new** ArithmeticException("\nNumber is negative, cannot calculate square");
6. }
7. **else** {
8. System.out.println("Square of " + num + " is " + (num\*num));
9. }
10. }
11. //main method
12. **public** **static** **void** main(String[] args) {
13. TestThrow obj = **new** TestThrow();
14. obj.checkNum(-3);
15. System.out.println("Rest of the code..");
16. }
17. }

**Output:**

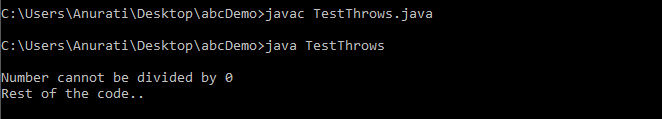


Java throws Example

**TestThrows.java**

1. **public** **class** TestThrows {
2. //defining a method
3. **public** **static** **int** divideNum(**int** m, **int** n) **throws** ArithmeticException {
4. **int** div = m / n;
5. **return** div;
6. }
7. //main method
8. **public** **static** **void** main(String[] args) {
9. TestThrows obj = **new** TestThrows();
10. **try** {
11. System.out.println(obj.divideNum(45, 0));
12. }
13. **catch** (ArithmeticException e){
14. System.out.println("\nNumber cannot be divided by 0");
15. }
17. System.out.println("Rest of the code..");
18. }
19. }

**Output:**



Access Modifier

* **public:** Accessible in all classes in your application.
* **protected:** Accessible within the package in which it is defined and in its **subclass(es) (including subclasses declared outside the package)**.
* **private:** Accessible only within the class in which it is defined.
* **default (declared/defined without using any modifier):** Accessible within the same class and package within which its class is defined.

What is Oops

Object-oriented programming (OOP) is **a way of thinking about and organizing code for maximum reusability**. With this type of programming, a program comprises objects that can interact with the user, other objects, or other programs. This makes programs more efficient and easier to understand.

Pillars of OOPs

* [Abstraction](https://www.geeksforgeeks.org/abstraction-in-java-2/)
* [Encapsulation](https://www.geeksforgeeks.org/encapsulation-in-java/)
* [Inheritance](https://www.geeksforgeeks.org/inheritance-in-java/)
* [Polymorphism](https://www.geeksforgeeks.org/polymorphism-in-java/)

What is Abstraction

Hiding internal implementation just high let set of service what we are offering.

Example – ATM Machine

**Advantages of Abstraction**

1. It reduces the complexity of viewing things.
2. Avoids code duplication and increases reusability.
3. Helps to increase the security of an application or program as only essential details are provided to the user.
4. It improves the maintainability of the application.
5. It improves the modularity of the application.
6. The enhancement will become very easy because without affecting end-users we can be able to perform any type of changes in our internal system

What is Encapsulation

the process of grouping data members and corresponding method into a single unit it is called encapsulation

**Advantages of Encapsulation**:

* **Data Hiding:**it is a way of restricting the access of our data members by hiding the implementation details. Encapsulation also provides a way for data hiding. The user will have no idea about the inner implementation of the class. It will not be visible to the user how the class is storing values in the variables. The user will only know that we are passing the values to a setter method and variables are getting initialized with that value.
* **Increased Flexibility:** We can make the variables of the class read-only or write-only depending on our requirement. If we wish to make the variables read-only then we have to omit the setter methods like setName(), setAge(), etc. from the above program or if we wish to make the variables write-only then we have to omit the get methods like getName(), getAge(), etc. from the above program
* **Reusability:** Encapsulation also improves the re-usability and is easy to change with new requirements.
* **Testing code is easy:** Encapsulated code is easy to test for unit testing.

What is Inheritance

inheritance, a class (Sub Class) can inherit properties of another class (Super Class). Sub class can have its own properties along with the inherited properties from its super class.

Advantage

* **Code Reusability:**The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.

## ****How to use inheritance in Java?****

The **extends keyword**is used for inheritance in java. Using the extends keyword indicates you are derived from an existing class. In other words, “extends” refers to increased functionality

What is Polymorphism

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

**Real-life Illustration:**Polymorphism

A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person possesses different behavior in different situations. This is called polymorphism.

* Compile-time Polymorphism
* Runtime Polymorphism

**Method Overloading**: When there are multiple functions with the same name but different parameters then these functions are said to be **overloaded** Static polymorphism or Compile time Polymorphism

**class** Helper {

    // Method with 2 integer parameters

**static** **int** Multiply(**int** a, **int** b)

    {

        // Returns product of integer numbers

**return** a \* b;

    }

    // Method 2

    // With same name but with 2 double parameters

**static** **double** Multiply(**double** a, **double** b)

    {

        // Returns product of double numbers

**return** a \* b;

    }

}

// Class 2

// Main class

**class** GFG {

    // Main driver method

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

    {

        // Calling method by passing

        // input as in arguments

        System.out.println(Helper.Multiply(2, 4));

        System.out.println(Helper.Multiply(5.5, 6.3));

    }

}

Method Overriding

It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding. [**Method overriding**](https://www.geeksforgeeks.org/overriding-in-java/) Runtime Polymorphism Dynamic Poly…

**class** Parent {

    // Method of parent class

**void** Print()

    {

        // Print statement

        System.out.println("parent class");

    }

}

// Class 2

// Helper class

**class** subclass1 **extends** Parent {

    // Method

**void** Print() { System.out.println("subclass1"); }

}

// Class 3

// Helper class

**class** subclass2 **extends** Parent {

    // Method

**void** Print()

    {

        // Print statement

        System.out.println("subclass2");

    }

}

// Class 4

// Main class

**class** GFG {

    // Main driver method

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

    {

        // Creating object of class 1

        Parent a;

        // Now we will be calling print methods

        // inside main() method

        a = **new** subclass1();

        a.Print();

        a = **new** subclass2();

        a.Print();

    }

}

# **Difference between abstract class and interface**

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

What is Interface

An **interface in Java** is a blueprint of a class. The interface in Java is a mechanism to achieve [*abstraction*](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java).