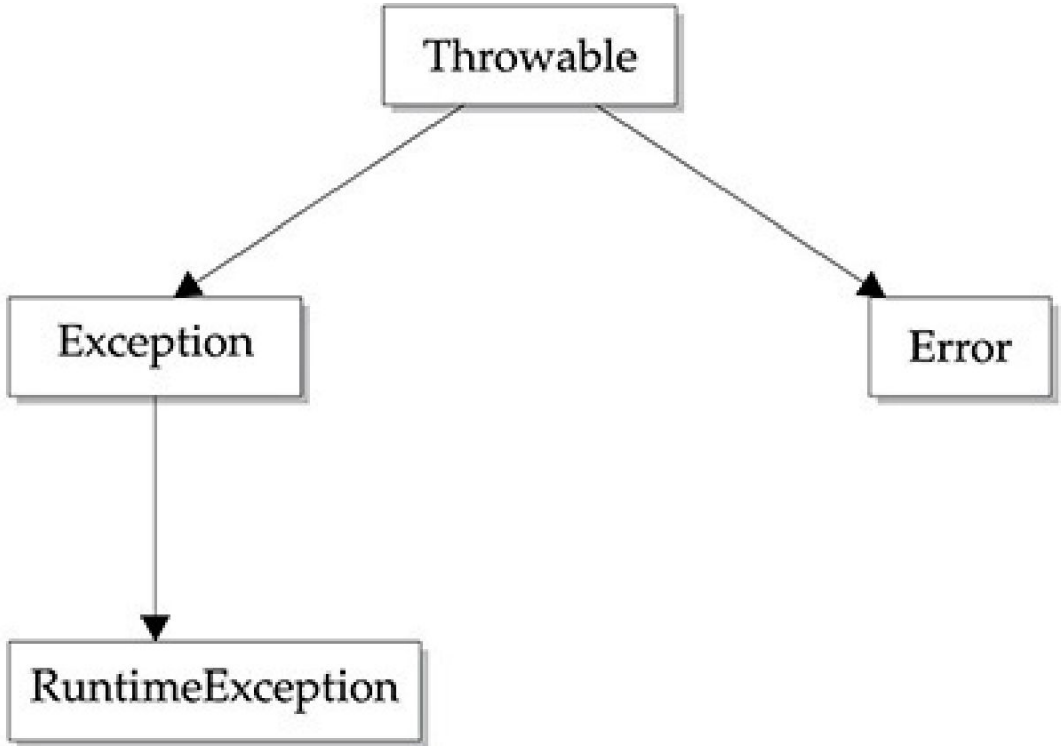
UNIT - II

Exception Handling in Java

Exception Handling in Java

- A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code.
- When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error.
- That method may choose to handle the exception itself, or pass it on.
- Either way, at some point, the exception is caught and processed.
- Exceptions can be generated by the Java run-time system, or they can be manually generated by your code.

Exception Types



try-catch mechanism

```
class Excep1
{
	public static void main(String args[])
	{
	int d, a;

		try
		{
		d = 0;
		a = 42/1;
		System.out.println("This is not error");
		}catch (ArithmeticException e)
			{
			 System.out.println("Error due to Division by Zero");
			}


	}
```

```
class Excep1
{
	public static void main(String args[])
	{
	int d, a;

		try
		{
		d = 0;
		a = 42/d;
		System.out.println("This is not error");
		}catch (ArithmeticException e)
			{
			 System.out.println("Error due to Division by Zero");
			}


	}
```

throw statement

- "throw" statement to handle errors related to patient weight in a medical context:

"throw" statement - to handle errors related to patient weight

```
import java.util.Scanner;

public class MedicationDosageCalculator {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter patient weight in kg: ");
        double weight = input.nextDouble();

        try {
            double dosage = calculateDosage(weight);
            System.out.println("The recommended medication dosage is " + dosage + "mg.");
        } catch (IllegalArgumentException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }

public static double calculateDosage(double weight) {
        if (weight < 0) {
            throw new IllegalArgumentException("Weight cannot be negative.");
        }

        // Calculate dosage based on weight
        double dosage = weight * 10;

        return dosage;
    }

}
```

throws statement

- In Java, the throws keyword is used to declare that a method may **throw a specific exception.**
- This allows the calling method to handle the exception appropriately.

throws statement

```
import java.util.Scanner;

public class VaccinationEligibilityChecker {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter patient age: ");
        int age = input.nextInt();

      try {
            boolean eligible = checkEligibility(age);
            if (eligible) {
                System.out.println("The patient is eligible for vaccination.");
            } else {
                System.out.println("The patient is not eligible for vaccination.");
            }
        } catch (AgeOutOfRangeException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
```

```
public static boolean checkEligibility(int age) throws AgeOutOfRangeException {
        if (age < 0 || age > 120) {
            throw new AgeOutOfRangeException("Invalid age: " + age);
        }

        // Check eligibility based on age
        boolean eligible = false;
        if (age >= 18) {
            eligible = true;
        }

        return eligible;
    }

}

class AgeOutOfRangeException extends Exception {
    public AgeOutOfRangeException(String message) {
        super(message);
    }
}
```

Finally

- In Java, the finally block is used to define code that should be executed regardless of whether an exception is thrown or not

Finally

```
import java.util.Scanner;

public class BloodPressureMonitor {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter patient blood pressure (systolic/diastolic): ");
        int systolic = input.nextInt();
        int diastolic = input.nextInt();

            try {
            boolean normal = checkBloodPressure(systolic, diastolic);
            if (normal) {
                System.out.println("The patient's blood pressure is normal.");
            } else {
                System.out.println("The patient's blood pressure is high.");
            }
        } catch (IllegalArgumentException e) {
            System.err.println("Error: " + e.getMessage());
        } finally {
            System.out.println("Thank you for using the blood pressure monitor.");
        }
    }
```

```
public static boolean checkBloodPressure(int systolic, int diastolic) throws
IllegalArgumentException {
        if (systolic <= 0 || diastolic <= 0) {
            throw new IllegalArgumentException("Blood pressure values must be positive.");
        }
        if (systolic < 90 || systolic > 120 || diastolic < 60 || diastolic > 80) {
            return false; // blood pressure is high
        }
        return true; // blood pressure is normal
    }

}
```

Built-in-Exceptions

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalCallerException	A method cannot be legally executed by the calling code.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
LayerInstantiationException	A module layer cannot be created.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBoundsException	Attempt to index outside the bounds of a string.
TypeNotPresentException	Type not found.
UnsupportedOperationException	An unsupported operation was encountered.

Custom Exception

Method	Description
final void addSuppressed(Throwable *exc*)	Adds *exc* to the list of suppressed exceptions associated with the invoking exception. Primarily for use by the **try**-with-resources statement.
Throwable fillInStackTrace( )	Returns a **Throwable** object that contains a completed stack trace. This object can be rethrown.
Throwable getCause( )	Returns the exception that underlies the current exception. If there is no underlying exception, **null** is returned.
String getLocalizedMessage( )	Returns a localized description of the exception.
String getMessage( )	Returns a description of the exception.
StackTraceElement[ ] getStackTrace( )	Returns an array that contains the stack trace, one element at a time, as an array of **StackTraceElement**. The method at the top of the stack is the last method called before the exception was thrown. This method is found in the first element of the array. The **StackTraceElement** class gives your program access to information about each element in the trace, such as its method name.
final Throwable[ ] getSuppressed( )	Obtains the suppressed exceptions associated with the invoking exception and returns an array that contains the result. Suppressed exceptions are primarily generated by the **try**-with-resources statement.
Throwable initCause(Throwable *causeExc*)	Associates *causeExc* with the invoking exception as a cause of the invoking exception. Returns a reference to the exception.
void printStackTrace( )	Displays the stack trace.
void printStackTrace(PrintStream *stream*)	Sends the stack trace to the specified stream.
void printStackTrace(PrintWriter *stream*)	Sends the stack trace to the specified stream.
void setStackTrace(StackTraceElement *elements*[ ])	Sets the stack trace to the elements passed in *elements*. This method is for specialized applications, not normal use.
String toString( )	Returns a **String** object containing a description of the exception. This method is called by **println( )** when outputting a **Throwable** object.