# Multithreading, Exception Handling, and OOPs

# You Already Know

**Course(s):**

Core Java

# Recap

- Explain polymorphism and inheritance
  - Method overloading
  - Abstract classes

- Explain the types of exceptions
  - *Try-catch*
  - *Finally* implementation
  - Exception types

- Explain the advantages of exception handling
  - Custom exceptions

# Recap

- Explain file handling
  - Create
  - Read
  - Modify

- Explain serialization and deserialization
  - Serialization
  - Deserialization

# A Day in the Life of a Full Stack Developer

It’s another day in Joe’s life as a Full Stack Developer in Abq Inc. Joe is appreciated for his progress and commitment. He seeks more challenging work to test his ability to check if he can perform the tasks within the stipulated time. Upon discussing with the Scrum Master and the team, he picks up a feature and assigns a task to himself on JIRA.

He decides to develop functionality that creates a file in the application to store all the errors, warnings, and exceptions which are raised when users work with the website. This file can be further reused to fix bugs and help the developers improve the performance of the application and provide a hassle-free experience to their users.

In this lesson, we will learn how to solve this real-world scenario to help Joe complete his task effectively and quickly.

# Learning Objectives

By the end of this lesson, you will be able to:

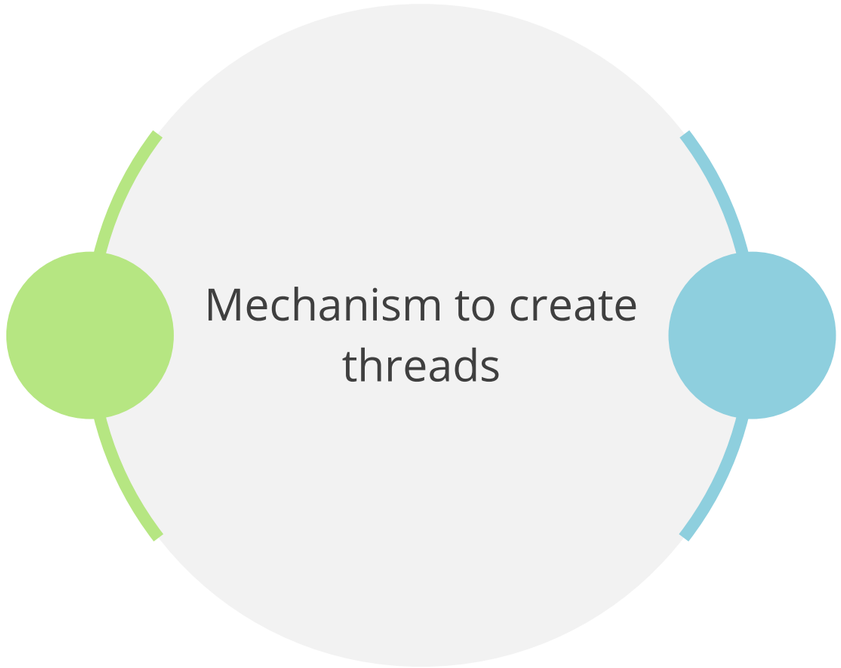
- Implement thread safety in the project
- Implement the custom exception handlers
- Enumerate the OOPs concepts in detail
- Create and perform CRUD operations on a file

# Concurrency and Multithreading

# Overview of Concurrency and Multithreading

- Concurrency is the ability to run multiple programs in parallel. It enables a program to achieve high performance and throughput by utilizing the untapped capabilities of the underlying operating system.

- Multithreading feature allows concurrent execution of multiple parts of a program for maximum CPU utilization. Each part of such a program is called a thread.




Extending the “Thread” class

Implementing the “Runnable” interface

# Mechanisms of Thread Creation

Extending the "Thread" class

```
class thrd extends Thread {

    public void run() {
        try { //some code }
        catch (Exception e) {//some code}
    }
}


  public class Thread {

      public static void main(String[] args) {

  thrd object = new thrd();
  object.start();
      }
  }
```

Implementing the "Runnable" interface

```
class thrd implements runnable {

    public void run() {
        try { //some code }
        catch (Exception e) {//some code}
    }
}


public class Thread {

    public static void main(String[] args) {

Thread object = new Thread(new thrd());
object.start();
    }
}
```

# Thread Safety

Thread safety is a process that makes the program safe to be used in a multithreaded environment.

Different ways to implement thread safety are listed below:

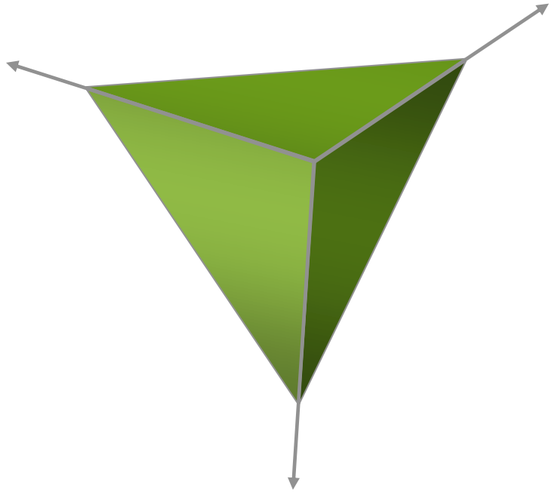
- Use of synchronization
- Use of atomic wrapper classes from *java.util.concurrent.atomic* package
- Use of locks from *java.util.concurrent.locks* package
- Use of thread safe collection classes
- Use of a volatile keyword with variables

# The wait(), notify(), and notifyAll()

The object class in Java has three final methods that allow threads to communicate about the locked status of a resource.

**wait()**

It intimates the calling thread to halt the lock and sleep until some other thread enters the same monitor and calls notify().



**notify()**

It wakes up a thread that called wait() on the same object. It intimates the waiting thread to wake up. It does not give up a lock on a resource.

**notifyAll()**

It wakes up all the threads that called wait() on the same object. The highest priority thread executes in most of the situations provided.

# The wait(), notify(), and notifyAll()

**Wait()**

```
Synchronized( lockObject ){

        while( !condition ) {

            lockObject.wait();
        }

// implementation goes here

}
```

**notify()**

```
Synchronized( lockObject ){

//implementation goes here

        lockObject.notify();


}
```

**notifyAll()**

```
Synchronized( lockObject ){

//implementation goes here

        lockObject.notifyAll();


}
```

# Programming the Threads Creation

**Duration: 45 min.**

**Problem Statement:**

You are given a project to demonstrate the implementation of thread creation mechanisms.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to demonstrate the implementation of thread creation mechanisms:

1. Create a Java project in your IDE
2. Write a program in Java to create a thread by extending the “Thread” class
3. Write another program in Java to create a thread by implementing the “Runnable” interface
4. Initialize the .git file
5. Add and commit the program files
6. Push the code to your GitHub repositories

# Demonstrate the Execution of sleep() and wait()

**Duration: 25 min.**

**Problem Statement:**

You are given a project to demonstrate the execution of sleep(), wait(), and its uses in the threading concept.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

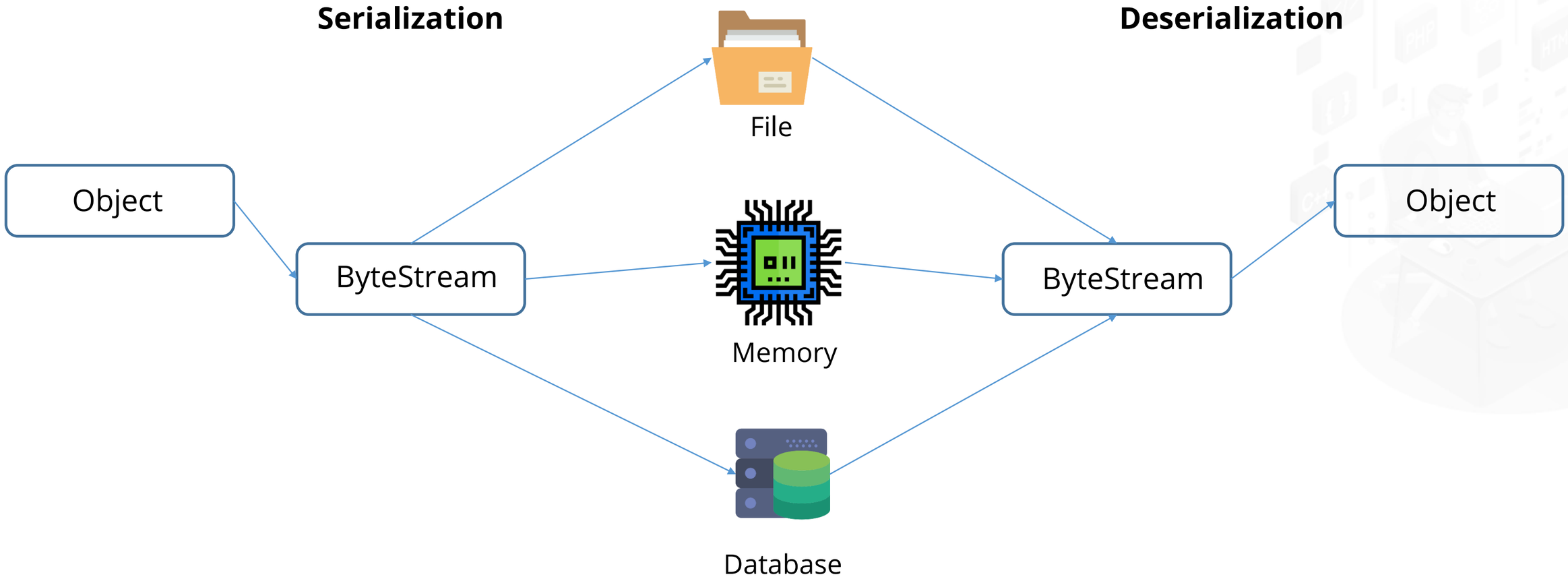
Steps to demonstrate the execution of sleep() and wait():

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate the execution of sleep() and wait()
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories

# Serialization and Deserialization

# Serialization and Deserialization

Serialization is a process of converting the state of an object into a byte stream. Deserialization is a reverse process of converting the byte stream to recreate the actual Java object in the memory.



# Thread Synchronization Mechanisms

**Duration: 35 min.**

**Problem Statement:**

You are given a project to demonstrate the multithreading with and without synchronization.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to demonstrate thread synchronization mechanisms:

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate the synchronization mechanisms
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories

# Exception Handling

# Types of Exceptions

## Asynchronous Exceptions

Deal with hardware and external problems

- Mouse failure
- Keyboard, motherboard failure
- Memory problems
- Power failure

java.lang.Error is a super class of all asynchronous exceptions

## Synchronous Exceptions

Deal with programmatic run-time errors

Types:

- Checked exceptions
- Unchecked exceptions

# Types of Synchronous Exceptions

## Checked Exceptions

Arise from external conditions during compile time and are expected to be handled by the programmer

Examples: “requested file not found” and “network failure”

## Unchecked Exceptions

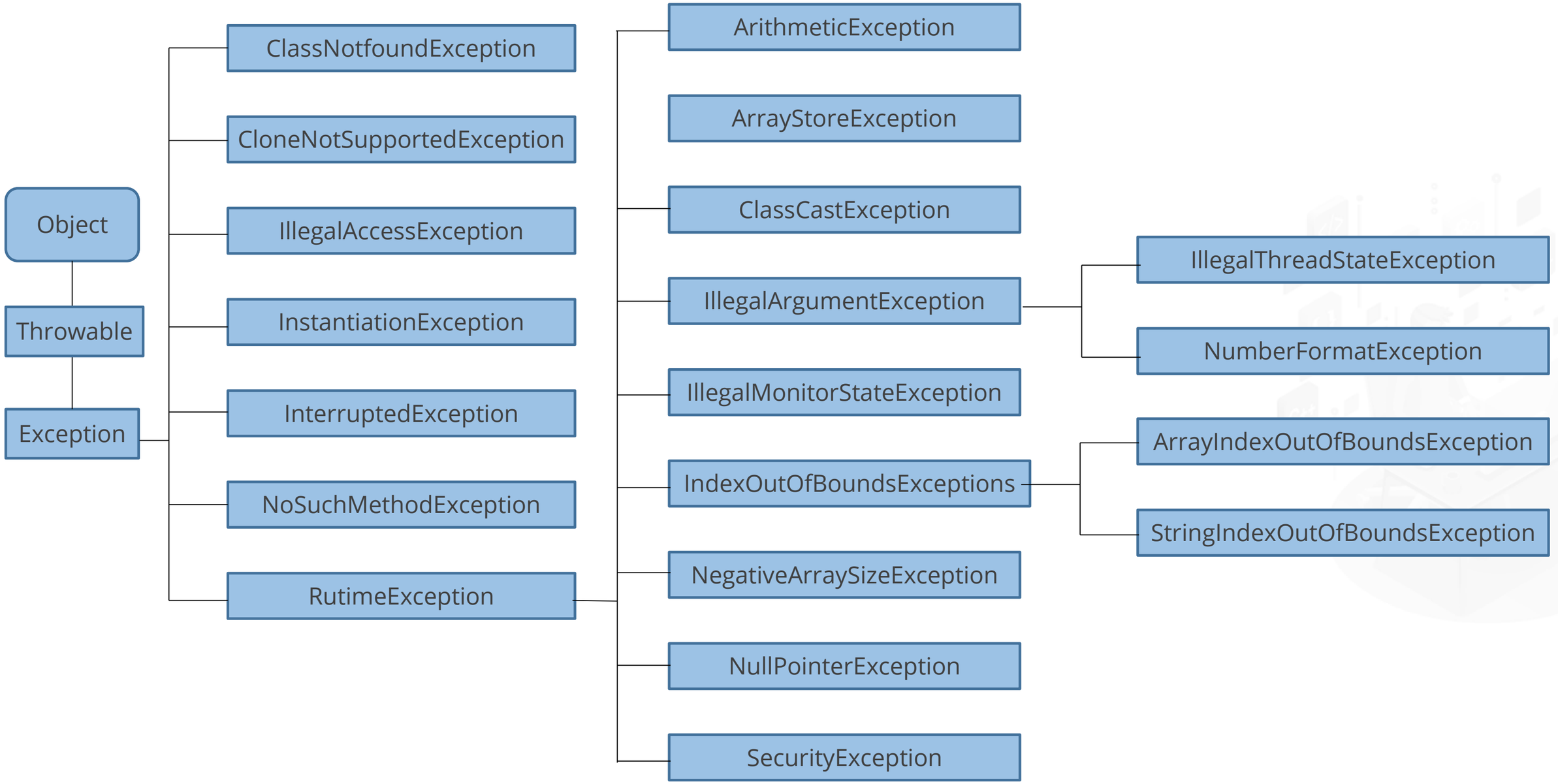
Arise from conditions that represent bugs during execution time or situations that are considered difficult to handle

Example: “runtime exceptions are attempting to access beyond the end of an array”

# Difference between Error Class and Exception Class

| Errors | Exceptions |
|---|---|
| Errors in Java are of type java.lang.Error. | Exceptions in Java are of type java.lang.Exception. |
| All errors in Java are of unchecked type. | Exceptions may be checked as well as unchecked. |
| Errors happen at run-time. They will not be known to the compiler. | Checked exceptions are known to compiler, whereas unchecked exceptions are not known to compiler because they occur at run-time. |
| It is impossible to recover from errors. | You can recover from exceptions by handling them through try-catch blocks. |
| Errors are mostly caused by the environment in which the application is running. | Exceptions are mainly caused by the application itself. |
| Examples :<br>java.lang.StackOverflowError, java.lang.OutOfMemoryError | Examples :<br>Checked Exceptions: SQLException, IOException<br>Unchecked Exceptions: ArrayIndexOutOfBoundException, ClassCastException, NullPointerException |

# Hierarchy of Exception



# *try-catch* Statement

When you write a Java program, you come across exceptions. In order to catch these exceptions, you have a *try-catch* statement.

Example of a program that handles exception:

```
Public class AddArguments2
{
  public static void main(String args[]) {
    try {
              int sum = 0;
              for (int i=0; i < args.length; i++) {
                        sum += Integer.parseInt (args [i]);
              }
              System.out.println("Sum = " +sum);
     } catch (NumberFormatException nfe) {
               system.err.println("One of the command-line" + "arguments is not an integer.");
     }
   }
}
```

# *try-catch* Statements

**Duration: 30 min.**

**Problem Statement:**

You are given a project to demonstrate the uses of *try-catch* blocks in Java.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to demonstrate the uses of try-catch blocks:

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate the execution of *try-catch* blocks with a real world scenario
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories

# Keywords and Custom Exceptions

**Duration: 30 min.**

**Problem Statement:**

You are given a project to demonstrate the uses of the throws, throw, finally, and custom exceptions in Java.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to demonstrate the uses of keywords and custom exceptions:

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate the uses of keywords in exceptions
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories

# Implement the Exception Handlers

**Duration: 30 min.**

**Problem Statement:**

You are given a project to demonstrate a program implementing the concept of exception handling available in Java and the custom exception handlers.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to implement exception handlers:

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate the uses of keywords in exceptions
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories

# File Handling

# The *StandardOpenOptions* Enums

| | |
|---|---|
| *WRITE* | – Opens the file for write access |
| *APPEND* | – Appends the new data to the end of the file |
| *TRUNCATE_EXISTING* | – Truncates the file to zero bytes |
| *CREATE_NEW* | – Creates a new file and throws an exception if the file already exists |
| *CREATE* | – Opens the file if it exists or creates a new file if it does not |
| *DELETE_ON_CLOSE* | – Deletes the file when the stream is closed |
| *SPARSE* | – Hints that a newly created file will be sparse |
| *SYNC* | – Keeps the file synchronized with the underlying storage device |

# Java Files

The *File* class provides many useful methods for creating and getting information about files.

| Method | Type | Function |
|---|---|---|
| canRead() | Boolean | Tests the file if it is readable or not |
| canWrite() | Boolean | Tests the file if it is writable or not |
| createNewFile() | Boolean | Creates an empty file |
| delete() | Boolean | Deletes a file |
| exists() | Boolean | Tests the file if it exists |
| getName() | String | Returns the name of the file |
| getAbsolutePath() | String | Returns the absolute pathname |
| length() | Long | Returns the file size in bytes |
| list() | String | Returns an array of the files in the directory |
| mkdir() | Boolean | Creates a directory |

# File Handling Mechanisms

**Duration: 30 min.**

**Problem Statement:**

You are given a project to demonstrate the create, read, update, and delete operations on the files in Java.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to demonstrate the file handling mechanisms:

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate the Create, Read, Update, and Delete operations on a file
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories

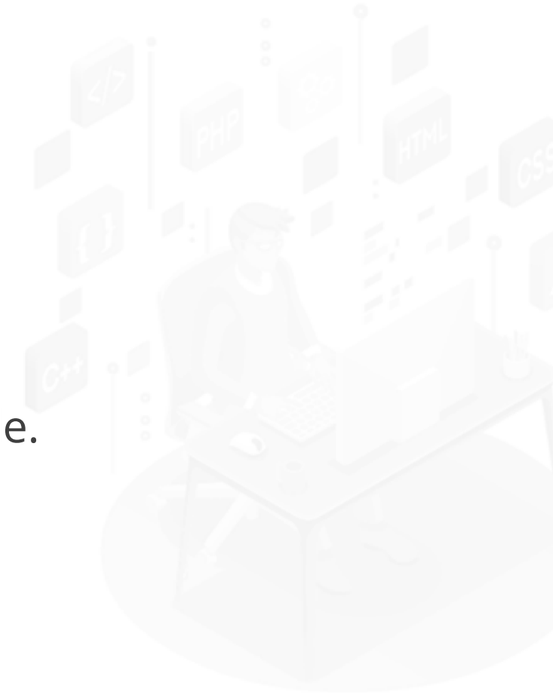
# Classes, Objects, and Pillars of OOPs

# Classes and Objects

A *class* is a building block and a template that describes the data and behavior associated with *instances* of a class. An entity that has a state and behavior is known as an *object*.

**Source file declaration rules:**

- A source file can have only one public class.
- It can contain multiple non-public classes.
- The public class name should be the name of the source file.
- The package statement should be the first statement in the source file.
- Import and package statements will imply to all the classes present in the source file.



# Encapsulation

Encapsulation is one the fundamental OOP concepts. It is a process of wrapping code and data together into a single entity.

**Encapsulation can be achieved by:**

- Declaring the variables of a class as *private.*
- Providing *public* setter and getter methods to modify and view the variable values.



**Advantages of Encapsulation:**

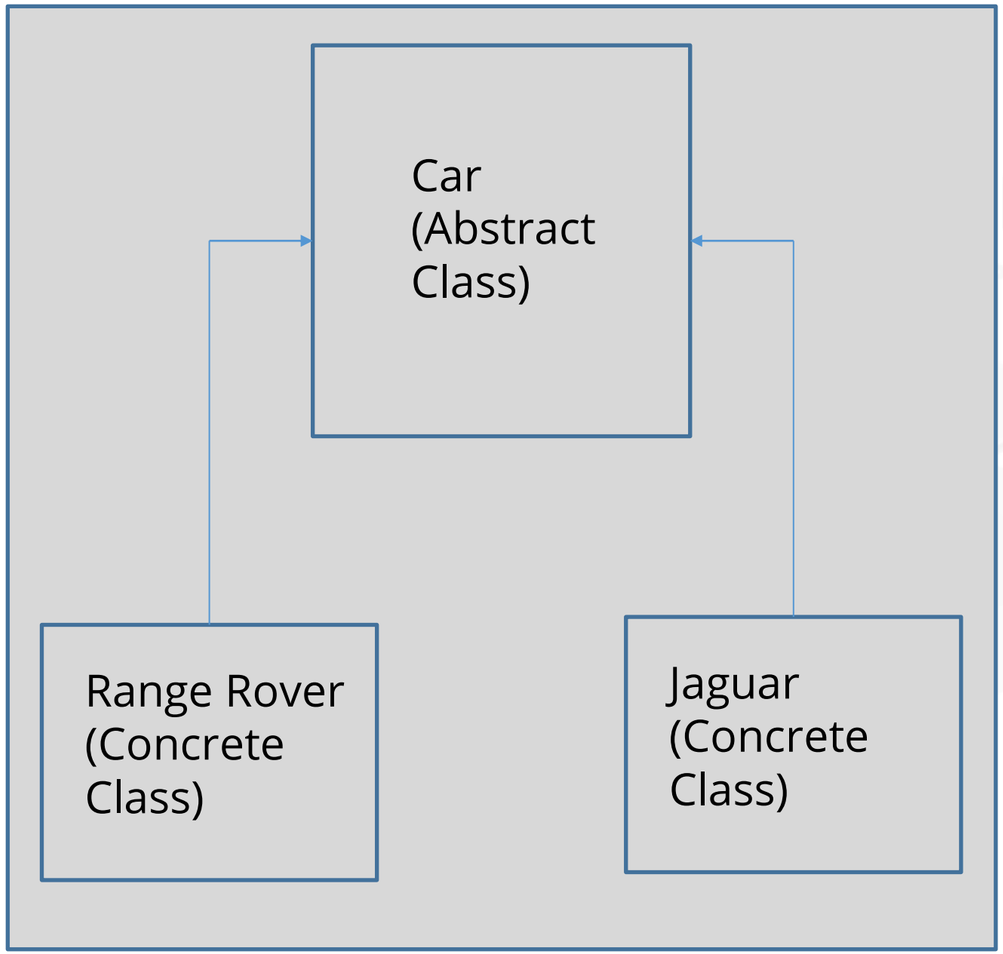
- It improves maintainability, flexibility, and reusability.
- The fields can be made read-only or write-only.

# Abstraction

Abstraction is a process of hiding the implementation details from users. Only the functionality is provided to the user.

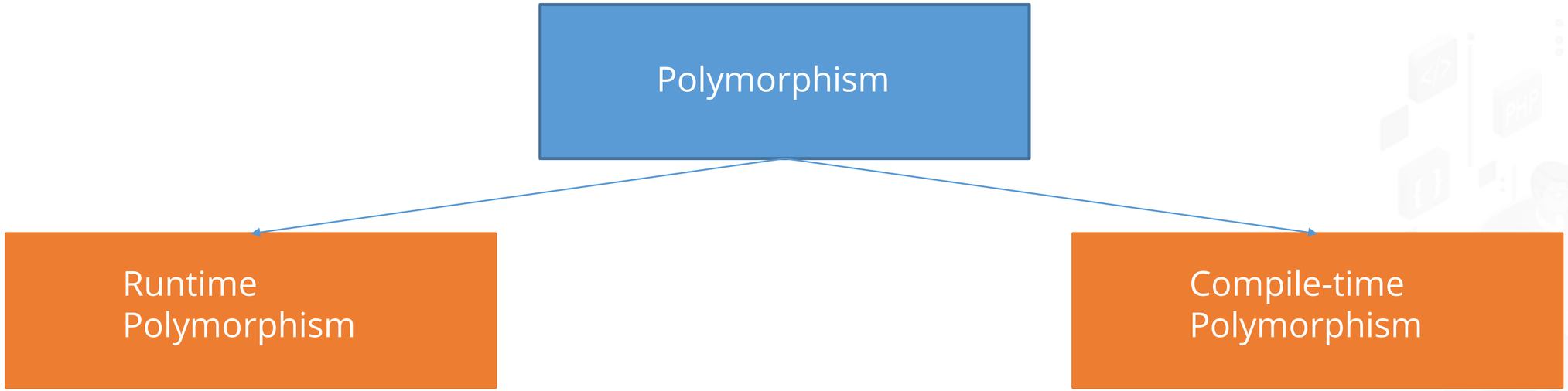
**Abstraction in Java**:

- Abstract class is a class that is declared with *abstract* keyword.
- Any class that contains more than one abstract method must be declared with *abstract* keyword.
- Abstract classes can't be directly instantiated with the *new* operator.
- An abstract class can have parameterized constructors. The default constructor is present in an abstract class.



# Polymorphism

Polymorphism is the ability of an object to perform a single action in different ways. The common use of polymorphism in OOP occurs when a parent class is used to refer to a child class object.




- It is achieved by **virtual functions** and **pointers**.
- It is more flexible as all programs execute at run time.

- It is achieved by **function** overloading and **operator** overloading.
- Less flexible as all the programs execute at compile time.

# Inheritance

Inheritance is a mechanism in which one class acquires all the properties and behaviors of the parent class.

It can be used for method overriding and code reusability.

Syntax of Java Inheritance:

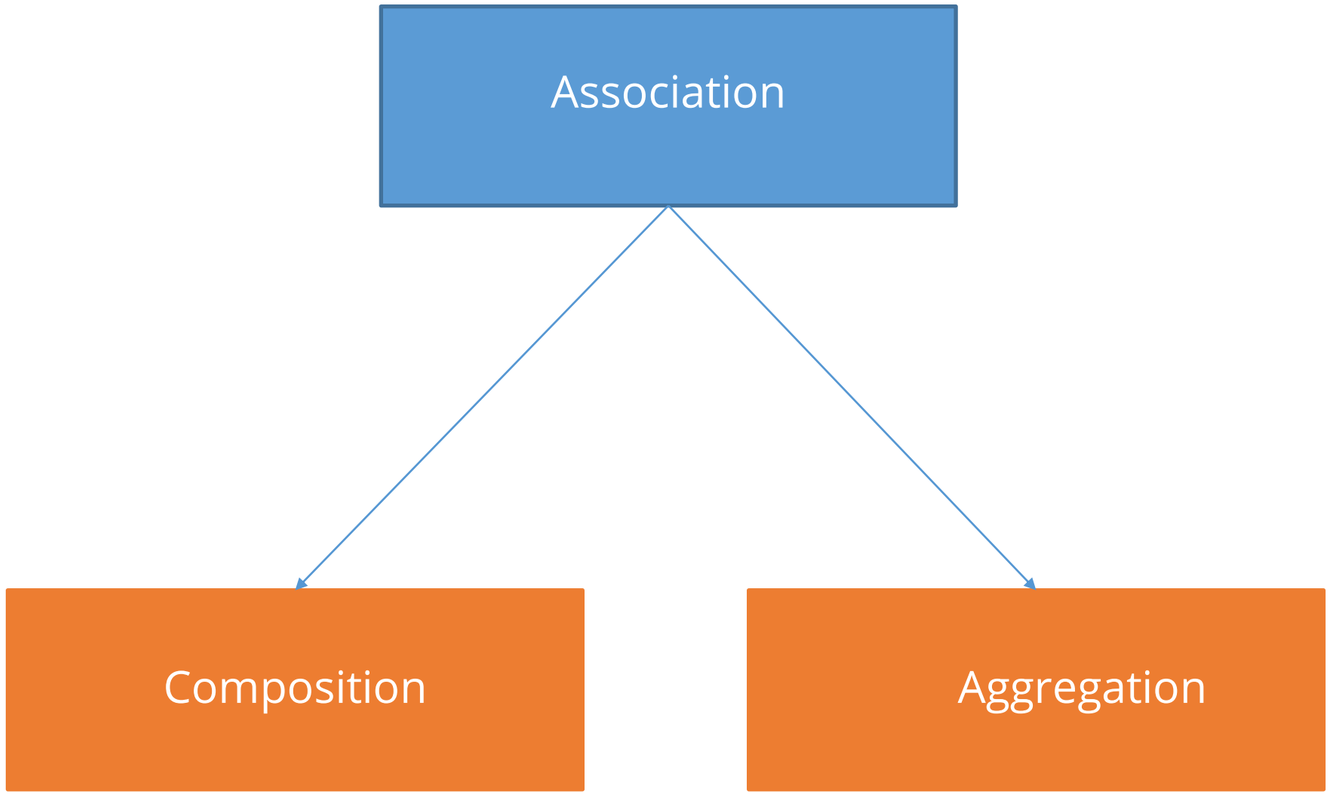
```
class Subclass-name extends Superclass-name
{
 //methods and fields
}
```

Indicates that you are creating a new class that is derived from an existing class

You can create new classes that are built upon existing classes using the inheritance feature in Java.

# Association, Aggregation, and Composition

Association is the relationship between multiple objects.




## Aggregation vs. Composition

- Dependency: Aggregation implies a relationship where the child **can exist independently** of the parent. Composition implies a relation where the child **cannot exist independently** of the parent.
- Type of Relationship: Aggregation is a "**has-a**" relation, whereas composition is a "**part-of**" relation.
- Type of Association: Composition is a **strong** association where as aggregation is a **weak** association.

# Class, Objects, and Pillars of OOPs

**Duration: 40 min.**

**Problem Statement:**

You are given a project to demonstrate the uses of classes, objects, and the object-oriented pillars in Java.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to demonstrate uses of classes and objects:

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate the uses of class, objects, and pillars of OOPs
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories

# The Diamond Problem

**Duration: 20 min.**

**Problem Statement:**

You are given a project to work and resolve the diamond problem using OOPs concepts.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to solve the diamond problem:

1. Create a Java project in your IDE
2. Write a program in Java to demonstrate the uses of class, objects, and pillars of OOPs
3. Initialize the .git file
4. Add and commit the program files
5. Push the code to your GitHub repositories

# Key Takeaways

- Multithreading is a process of executing multiple threads simultaneously.
- Exception handling is used to handle run-time errors to maintain normal flow of an application.
- Java is an object-oriented programming paradigm which provides several concepts like inheritance, polymorphism, data binding, encapsulation, and abstraction.
- Java I/O is used to process input and output data.

# File Handling

**Duration: 30 min.**

**Problem Statement:**
Write a program to read, write, and append a file.

LESSON-END PROJECT

# Before the Next Class

**Course:**

- Beginning Java Data Structures and Algorithms
- Maven Tutorial - Manage Java Dependencies in 20 Steps

**You should be able to:**

- Explain the fundamentals of data structure
- Measure the algorithmic complexity with the Big O notation
- Explain the fundamentals of Maven
- Set up your first Maven project
- Build a jar