ItsRunTym

JAVA

70+ interview questions/answers

1. **What is Java, and what are its main features?**

Answer: Java is a high-level, object-oriented programming language known for its platform independence, robustness, security, and simplicity. Its main features include:

- Platform independence
- Object-oriented
- Robust and secure
- Multithreaded
- Architecture-neutral
- Portable

2. **Explain the difference between JDK, JRE, and JVM.**

Answer:

Feature	**JDK (Java Development Kit)**	**JRE (Java Runtime Environment)**	**JVM (Java Virtual Machine)**
Definition	Software development kit for Java development	Runtime environment for executing Java programs	Abstract computing machine for executing Java bytecode
Components	Includes JRE, compilers, and development tools	Includes libraries and JVM	Executes Java bytecode

Feature	JDK (Java Development Kit)	JRE (Java Runtime Environment)	JVM (Java Virtual Machine)
Usage	Used for developing, debugging, and monitoring Java applications	Used for running Java applications	Provides runtime environment for Java applications

3. **Describe the principles of Write Once Run Anywhere (WORA) in Java.**

Answer: Write Once Run Anywhere (WORA) is a fundamental principle of Java that ensures that compiled Java code (bytecode) can be executed on any platform with a compatible Java Virtual Machine (JVM) without modification. This is achieved by compiling Java source code into platform-independent bytecode, which is then interpreted by the JVM at runtime.

4. **How does memory management work in Java?**

Answer: Memory management in Java is handled automatically by the Java Virtual Machine (JVM) through a process called garbage collection. The JVM manages memory allocation and deallocation by dynamically allocating memory to objects created during program execution and reclaiming memory from objects that are no longer referenced. This helps prevent memory leaks and ensures efficient memory usage.

5. **Differentiate between stack and heap memory in Java.**

Answer:

Feature	Stack Memory	Heap Memory
Purpose	Used for method frames, local variables, method call stack	Used for storing objects and dynamically allocated memory
Thread Safety	Thread-safe	Not thread-safe
Size	Limited in size	Larger than stack memory

Feature	Stack Memory	Heap Memory
Lifetime	Short-lived (variables are destroyed when the method completes)	Long-lived (objects persist until they are no longer referenced)

6. **Explain the difference between primitive and reference data types in Java.**

Answer: Primitive data types represent basic values and are stored directly in memory, whereas reference data types refer to objects stored on the heap and are accessed via reference variables. Primitive data types are passed by value, whereas reference data types are passed by reference.

7. **What is the significance of the main() method in Java?**

Answer: The **main()** method is the entry point of a Java program. It is where the execution of the program begins. The JVM looks for the **main()** method with the following signature to start the execution of the program:

public static void main(String[] args)

The **args** parameter allows command-line arguments to be passed to the program.

8. **List the different access specifiers in Java and their significance.**

Answer: Java provides four access specifiers:

- **public**: Accessible from anywhere.
- **protected**: Accessible within the same package or subclasses.
- **default** (no modifier): Accessible within the same package.
- **private**: Accessible only within the same class.

9. **Compare == and .equals() method in Java.**

Answer:

- **==**: Compares object references. It checks if two references point to the same memory location.
- **.equals()**: Compares the actual contents of objects. It is overridden by classes to define their own equality criteria.

10. **Explain the difference between ArrayList and LinkedList.**

Answer:

Feature	**ArrayList**	**LinkedList**
Data Structure	Resizable-array implementation	Doubly linked list implementation
Insertion/Deletion	Slower for insertions and deletions in the middle	Faster for insertions and deletions in the middle
Random Access	Faster random access	Slower random access
Memory Overhead	Less memory overhead	More memory overhead

11. **What is the static keyword used for in Java?**

Answer: The **static** keyword in Java is used to create variables and methods that belong to the class itself rather than instances of the class. Static variables (class variables) are shared among all instances of the class, while static methods can be called without creating an instance of the class.

12. **Define inheritance in Java with an example.**

Answer: Inheritance in Java allows a class (subclass) to inherit properties and behaviors (fields and methods) from another class (superclass). Here's an example:

```
// Superclass
class Animal {
    void eat() {
        System.out.println("Animal is eating...");
    }
}
// Subclass inheriting from Animal
class Dog extends Animal {
```

```
    void bark() {
        System.out.println("Dog is barking...");
    }
}
```

13. **Explain polymorphism in Java with an example.**

Answer: Polymorphism in Java allows objects of different classes to be treated as objects of a common superclass. It enables methods to be dynamically bound at runtime. There are two types of polymorphism in Java:

- **Compile-time polymorphism**: It is achieved using method overloading.
- **Runtime polymorphism**: It is achieved using method overriding.

Here's an example:

```
// Superclass
class Animal {
        void sound() {
                System.out.println("Animal makes a sound");
        }
}
// Subclasses
class Dog extends Animal {
        void sound() {
                System.out.println("Dog barks");
        }
}
class Cat extends Animal {
        void sound() {
                System.out.println("Cat meows");
```

```
    }
}
// Polymorphic method
public class Main {
    public static void main(String[] args) {
        Animal dog = new Dog();
        Animal cat = new Cat();
        dog.sound(); // Output: Dog barks
        cat.sound(); // Output: Cat meows
    }
}
```

14. **What is encapsulation in Java? Provide an example.**

Answer:
Encapsulation in Java is the mechanism of wrapping data (variables) and code (methods) together as a single unit (class) and restricting access to the internal state of the object.

It is achieved by declaring the variables of a class as private and providing public getter and setter methods to access and modify the data.

Example:

```
// Encapsulated class
class Person {
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
```

```
        this.name = name;
    }
}
// Usage
public class Main {
    public static void main(String[] args) {
        Person person = new Person();
        person.setName("John");
        System.out.println("Name: " + person.getName());
        // Output: Name: John
    }
}
```

15. **Describe abstraction in Java and provide an example.**

Answer:

Abstraction in Java is the process of hiding the implementation details and showing only the essential features of an object.

It allows the creation of abstract classes and interfaces with abstract methods. Abstraction is achieved using abstract classes and interfaces in Java.

Example:

```
// Abstract class
abstract class Shape {
    abstract void draw(); // Abstract method
}
// Concrete subclasses
class Circle extends Shape {
    void draw() {
```

```
            System.out.println("Drawing a circle");
        }
    }
    class Rectangle extends Shape {
        void draw() {
            System.out.println("Drawing a rectangle");
        }
    }
    // Usage
    public class Main {
        public static void main(String[] args) {
            Shape circle = new Circle();
            Shape rectangle = new Rectangle();
            circle.draw();    // Output: Drawing a circle
            rectangle.draw();    // Output: Drawing a rectangle
        }
    }
```

16. What are abstract classes and interfaces? How do they differ?

Answer:

Feature	**Abstract Class**	**Interface**
Definition	Class with one or more abstract methods	Collection of abstract methods and constants
Multiple Inheritance	Supports single inheritance and multiple inheritance	Supports multiple inheritance through interfaces
Implementation	Can have both abstract and concrete methods	All methods are abstract, no concrete implementation

Feature	Abstract Class	Interface
Constructor	Can have constructors	Cannot have constructors
Access Modifiers	Can have any access modifiers	All methods are implicitly public
Default Methods	Can have default methods in Java 8	Can have default methods in Java 8

17. **What is the final keyword signify in Java?**

Answer: The **final** keyword in Java is used to restrict the user from changing the value of a variable, from extending a class, or from overriding a method. Once a variable, method, or class is declared as **final**, its value or implementation cannot be modified.

18. **Differentiate between method overloading and method overriding.**

Answer:

Feature	Method Overloading	Method Overriding
Definition	Multiple methods with the same name but different signatures	Method in subclass with the same signature as in superclass
Syntax	Within the same class	Between superclass and subclass
Inheritance	Not dependent on inheritance	Dependent on inheritance
Return type	Can have different return types	Must have the same return type
Compile-time resolution	Resolved during compile-time based on method signature	Resolved during runtime based on object type
Example	**java public void add(int a, int b) {} java public void add(int a, int b, int c) {}**	**java class Animal { void sound() { } } java class Dog extends Animal { void sound() { } }**

19. **Explain the try-catch block in Java with an example.**

Answer: The **try-catch** block in Java is used for exception handling. Code that may throw an exception is enclosed within the **try** block, and the potential exceptions are caught and handled in the **catch** block.

Here's an example:

```
try {                       // Code that may throw an exception
    int result = 10 / 0;            // Division by zero
} catch (ArithmeticException e) {       // Exception handling
    System.out.println("Cannot divide by zero");
}
```

20. **What is the difference between checked and unchecked exceptions?**

Answer:

Feature	**Checked Exceptions**	**Unchecked Exceptions**
Definition	Subclass of **Exception** class, checked at compile-time	Subclass of **RuntimeException**, not checked at compile-time
Handling	Must be handled using try-catch or throws keyword	Can be handled but not required, typically programming errors
Examples	**FileNotFoundException**, **IOException**	**NullPointerException**, **ArrayIndexOutOfBoundsException**

21. **How are throw and throws keywords used in Java?**

Answer:

- **throw**: Used to explicitly throw an exception within a method or block of code.
- **throws**: Used in the method signature to indicate that the method may throw one or more exceptions, and the responsibility of handling those exceptions is delegated to the caller method or the JVM.

22. **Describe the purpose of the finally block in Java.**

Answer: The **finally** block in Java is used to ensure that a section of code is always executed, whether an exception is thrown or not. It is often used for releasing resources such as closing files or database connections. The **finally** block is executed after the **try** block (and any associated **catch** blocks) regardless of whether an exception is thrown or caught.

23. **Explain the use of the super keyword in Java.**

Answer:

- **super** keyword in Java is used to refer to the immediate parent class object.
- It is used to invoke superclass methods, access superclass variables, and invoke superclass constructors.
- It can be used to differentiate between superclass and subclass members with the same name.

24. **What is multithreading, and how is it implemented in Java?**

Answer:

- Multithreading in Java is a mechanism where multiple threads execute simultaneously within a single process to improve application performance.
- It allows concurrent execution of two or more parts of a program to maximize CPU utilization.
- In Java, multithreading can be implemented by extending the **Thread** class or implementing the **Runnable** interface.

```
class MyThread extends Thread {
   public void run() {
      for (int i = 1; i <= 5; i++) {
         System.out.println("Thread: " + i);
         try {
            Thread.sleep(1000); // Sleep for 1 second
         } catch (InterruptedException e) {
            System.out.println(e);
```

```
        }
    }
  }
}


public class Main {
  public static void main(String[] args) {
    MyThread t1 = new MyThread();
    MyThread t2 = new MyThread();
    t1.start();
    t2.start();
  }
}
```

25. **Why is synchronization important in Java?**

Answer:

- Synchronization in Java is important to prevent multiple threads from accessing shared resources concurrently, which can lead to data inconsistency and thread interference.
- It ensures that only one thread can access a shared resource at a time, preventing race conditions and maintaining data integrity.
- Synchronization is achieved using synchronized methods or blocks in Java.

26. **Define deadlock in Java and methods to prevent it.**

Answer:

- Deadlock in Java occurs when two or more threads are blocked indefinitely, waiting for each other to release resources that they need.

- Deadlocks can be prevented by following practices such as avoiding circular dependencies, using a consistent order when acquiring multiple locks, and setting a timeout for lock acquisition.

27. **What are the different ways to create threads in Java?**

Answer:

- Threads in Java can be created by extending the **Thread** class and overriding its **run()** method.
- They can also be created by implementing the **Runnable** interface and passing it to a **Thread** object.
- Java 8 introduced the **ExecutorService** framework for creating and managing threads using thread pools.

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("MyRunnable is running");
    }
}

public class Main {
    public static void main(String[] args) {
        Thread thread = new Thread(new MyRunnable());
        thread.start();
    }
}
```

28. **Explain the purpose of the volatile keyword in Java.**

Answer:

- The **volatile** keyword in Java is used to indicate that a variable's value may be changed by multiple threads simultaneously.

- It ensures that any thread reading the variable sees the most recent value written to it by any other thread, preventing thread caching of variables.

29. Describe generics in Java.

Answer:

Generics in Java allow types (classes and interfaces) to be parameterized by other types. They provide compile-time type safety by allowing classes, interfaces, and methods to operate on objects of specified types.

30. **Explain the Comparable interface in Java.**

Answer:

- The **Comparable** interface in Java is used to define the natural ordering of objects of a class.
- It contains a single method **compareTo()** which compares the current object with another object of the same type.
- Classes that implement **Comparable** can be sorted using methods like **Collections.sort()**.

Example: java // Sorting objects of type Integer using Comparable interface Collections.sort(listOfIntegers);

31. **Describe the Comparator interface in Java.**

Answer:

- The **Comparator** interface in Java is used to define custom ordering of objects.
- It contains a method **compare()** which compares two objects of the same type.
- It allows sorting of objects based on different criteria.

Example: java // Sorting objects of type String based on length using Comparator interface Collections.sort(listOfStrings, Comparator.comparing(String::length));

32. **What is the purpose of the toString() method in Java?**

Answer:

- The **toString()** method in Java is used to return a string representation of an object.
- It is automatically called when an object is concatenated with a string or passed to **System.out.println()**.
- It is often overridden in classes to provide meaningful string representations.

Example: java // Printing string representation of an object System.out.println(obj.toString());

33. **Explain the equals() method in Java.**

Answer:

- The **equals()** method in Java is used to compare the contents of two objects for equality.
- It is inherited from the **Object** class and is overridden in classes to define custom equality criteria.
- By default, **equals()** compares object references for equality.

Example: java // Comparing two objects for equality boolean isEqual = obj1.equals(obj2);

34. **Describe the hashCode() method in Java.**

Answer:

- The **hashCode()** method in Java is used to return a hash code value for an object.
- It is used in conjunction with hashing-based collections like **HashMap**, **HashSet**, etc.
- It is often overridden in classes that override the **equals()** method to ensure consistent behavior.

Example: java // Getting the hash code of an object int hash = obj.hashCode();

35. **Differentiate between StringBuilder and StringBuffer.**

Answer:

Feature	StringBuilder	StringBuffer
Thread Safety	Not thread-safe	Thread-safe
Performance	Faster	Slower
Synchronization	Not synchronized	Synchronized
Usage	Suitable for single-threaded environments	Suitable for multi-threaded environments

36. **Explain the difference between == and equals() method in Java.**

Answer:

- **==** operator is used to compare object references, i.e., it checks if two references point to the same memory location.
- **equals()** method is used to compare the contents of objects, i.e., it checks if the values of two objects are equal based on the overridden implementation.

Example: String str1 = new String("hello"); String str2 = new String("hello");

System.out.println(str1 == str2); // false (different memory locations)
System.out.println(str1.equals(str2)); // true (same content) ```

37. **What is an anonymous class in Java?**

Answer:

- An anonymous class in Java is a class without a name that is defined and instantiated at the same time.
- It is typically used for one-time use where creating a separate class is not justified.
- It is declared and instantiated in a single expression.

Example:
java Runnable r = new Runnable() {

public void run() { System.out.println("Anonymous class implementation");

}

};

38. **Describe the static block in Java.**

Answer:

- The **static** block in Java is a block of code enclosed within **{}** and preceded by the **static** keyword.
- It is executed only once when the class is loaded into memory by the JVM.
- It is often used for static initialization of class variables.

Example: java

class MyClass {

static {

System.out.println("Static block executed");

}

}

39. **What are Java annotations, and how are they used?**

Answer:

- Java annotations are metadata that provide data about a program but do not directly affect the program's execution.
- They are used for adding metadata information to Java source code, which can be processed at compile time or runtime.
- Annotations are declared using the **@** symbol followed by the annotation name.

Example: java @Override public void myMethod() { // Method implementation }

40. **Explain the purpose of the transient keyword in Java.**

Answer:

- The **transient** keyword in Java is used to indicate that a variable should not be serialized during object serialization.
- It is typically used for variables that hold temporary or non-essential data that does not need to be persisted.

Example: java class MyClass implements Serializable { private transient int tempData; // Not serialized private int regularData; // Serialized }

41. **Describe the purpose of the volatile keyword in Java.**

Answer:

- The **volatile** keyword in Java is used to indicate that a variable's value may be changed by multiple threads simultaneously.
- It ensures that any thread reading the variable sees the most recent value written to it by any other thread, preventing thread caching of variables.

Example: java class SharedResource { private volatile boolean flag; // Other members and methods }

42. **Explain the difference between static and final keywords in Java.**

Answer:

Feature	**static Keyword**	**final Keyword**
Definition	Class-level member, shared among all instances	Instance-level member, cannot be changed after initialization
Memory	Stored in method area (class memory)	Stored in heap or stack depending on usage
Modification	Can be modified	Cannot be modified after initialization
Initialization	Initialized only once when class is loaded	Initialized only once at the time of declaration
Example	**static int count;**	**final double PI = 3.14;**

43. **What is the purpose of the strictfp keyword in Java?**

Answer:

- The **strictfp** keyword in Java is used to restrict floating-point calculations to ensure portability.
- It ensures that floating-point calculations produce the same results across all platforms by adhering to IEEE 754 standards.

44. **Explain the instanceof operator in Java.**

Answer:

- The **instanceof** operator in Java is used to test if an object is an instance of a particular class or interface.
- It returns **true** if the object is an instance of the specified class or any of its subclasses, otherwise returns **false**.

45. **Describe the purpose of the assert keyword in Java.**

Answer:

- The **assert** keyword in Java is used to perform assertion checking, i.e., to verify assumptions about the program's state.
- It throws an **AssertionError** if the assertion condition is false, indicating that there is a logical error in the program.

46. **What is autoboxing and unboxing in Java?**

Answer:

- Autoboxing is the automatic conversion of primitive data types to their corresponding wrapper classes.
- Unboxing is the automatic conversion of wrapper class objects to their corresponding primitive data types.

47. **Explain the purpose of the Enum class in Java.**

Answer:

- The **Enum** class in Java is the base class for enumeration types.

- It provides methods for working with enum constants, such as iterating over them and converting between enum constants and their string representations.

48. **Describe the assert keyword in Java and its usage.**

Answer:

- The **assert** keyword in Java is used for assertion checking, i.e., to verify assumptions about the program's state.
- It throws an **AssertionError** if the assertion condition is false, indicating that there is a logical error in the program.
- Assertion checking can be enabled or disabled using JVM arguments **-ea** (enable assertions) and **-da** (disable assertions), respectively.

49. **Explain the purpose of the super() constructor call in Java.**

Answer:

- The **super()** constructor call in Java is used to invoke the constructor of the superclass from the subclass constructor.
- It is typically used to initialize inherited fields or perform superclass initialization tasks before subclass-specific initialization.

50. **What is the purpose of the this keyword in Java?**

Answer:

- The **this** keyword in Java is used to refer to the current instance of the class.
- It can be used to access instance variables, invoke methods, or invoke constructors of the current object.
- It is often used to disambiguate between instance variables and method parameters with the same name.

51. **Describe the purpose of the package keyword in Java.**

Answer:

- The **package** keyword in Java is used to declare a Java package, which is a group of related classes and interfaces.

- It provides a namespace for organizing and managing classes and interfaces, preventing naming conflicts.
- It is often the first non-comment statement in a Java source file, indicating the package to which the class or interface belongs.

52. **Explain the difference between throw and throws keywords in Java.**

Answer:

- **throw** keyword is used to explicitly throw an exception from a method or block of code.
- **throws** keyword is used in the method signature to indicate that the method may throw one or more exceptions, and the responsibility of handling those exceptions is delegated to the caller method or the JVM.

53. **Describe the purpose of the try-with-resources statement in Java.**

Answer:

- The **try-with-resources** statement in Java is used to automatically close resources that implement the **AutoCloseable** interface.
- It ensures that the resources are closed even if an exception is thrown during the execution of the try block.
- It simplifies resource management and reduces the likelihood of resource leaks.

54. **Explain the concept of method references in Java.**

Answer:

- Method references in Java provide a shorthand notation for lambda expressions to call a method.
- They allow you to refer to methods or constructors without invoking them.
- They are used to improve the readability of lambda expressions and make code more concise.

55. **Describe the purpose of the default keyword in Java interfaces.**

Answer:

- The **default** keyword in Java interfaces is used to define default method implementations.
- It allows interfaces to provide method implementations without requiring implementing classes to override them.
- It enables backward compatibility by adding new methods to interfaces without breaking existing implementations.

56. **Explain the concept of functional interfaces in Java.**

Answer:

- Functional interfaces in Java are interfaces that contain only one abstract method.
- They can have any number of default or static methods, but only one abstract method.
- Functional interfaces are used to represent lambda expressions or method references and facilitate functional programming in Java.

57. **Describe the purpose of the Stream API in Java.**

Answer:

- The **Stream** API in Java provides a fluent and functional approach for processing collections of data.
- It allows for declarative operations such as filter, map, reduce, and collect on collections of objects.
- Streams enable parallel processing of data, improving performance on multicore systems.

58. **Explain the concept of lambda expressions in Java.**

Answer:

- Lambda expressions in Java are anonymous functions that can be passed as arguments to methods or stored in variables.
- They provide a concise way to express instances of single-method interfaces (functional interfaces).
- Lambda expressions facilitate functional programming by enabling the use of functional interfaces.

59. **What is the purpose of the Optional class in Java?**

Answer:

- The **Optional** class in Java is used to represent an optional value, which may or may not contain a non-null value.
- It helps to avoid NullPointerExceptions by providing a way to handle null values more effectively.
- **Optional** encourages developers to explicitly handle the absence of a value, improving code clarity and robustness.

60. **Describe the purpose of the java.util.function package in Java.**

Answer:

- The **java.util.function** package in Java provides functional interfaces to represent lambda expressions or method references.
- It contains common functional interfaces such as **Predicate**, **Function**, **Consumer**, **Supplier**, etc.
- These interfaces are widely used in functional programming and with features like streams and lambda expressions.

61. **Explain the concept of default methods in Java interfaces.**

Answer:

- Default methods in Java interfaces are methods with a default implementation provided within the interface itself.
- They allow interfaces to provide method implementations without requiring implementing classes to override them.
- Default methods enable backward compatibility by adding new methods to interfaces without breaking existing implementations.

62. **What is the purpose of the static keyword in Java interfaces?**

Answer:

- The **static** keyword in Java interfaces is used to define static methods that can be called without an instance of the interface.
- Static methods in interfaces provide utility methods or factory methods related to the interface's functionality.

- They cannot be overridden by implementing classes and are not inherited by implementing classes.

63. **Explain the purpose of the java.lang.Math class in Java.**

Answer:

- The **java.lang.Math** class in Java provides methods for performing basic mathematical operations such as trigonometric, exponential, logarithmic, and rounding functions.
- It is a utility class and all its methods are static, meaning they can be called directly using the class name without creating an instance of the class.

64. **Describe the purpose of the java.util.Arrays class in Java.**

Answer:

- The **java.util.Arrays** class in Java provides static methods for working with arrays, such as sorting, searching, and manipulating arrays.
- It contains methods for sorting arrays using different sorting algorithms, searching for elements in arrays, and filling arrays with specific values.

65. **What is the purpose of the java.util.Collections class in Java?**

Answer:

- The **java.util.Collections** class in Java provides utility methods for working with collections, such as lists, sets, and maps.
- It contains methods for sorting, shuffling, searching, and synchronizing collections, as well as creating read-only or synchronized views of collections.

66. **Explain the purpose of the java.util.Comparator interface in Java.**

Answer:

- The **java.util.Comparator** interface in Java is used to define custom ordering of objects.
- It contains a method **compare()** which compares two objects of the same type.

- Comparator objects are often used to sort collections of objects that do not implement the **Comparable** interface or to provide alternative sorting criteria.

67. **Describe the purpose of the java.util.Iterator interface in Java.**

Answer:

- The **java.util.Iterator** interface in Java is used to iterate over elements of a collection sequentially.
- It provides methods for checking if there are more elements (**hasNext()**), retrieving the next element (**next()**), and removing the current element (**remove()**).
- Iterator objects are obtained from collection classes using the **iterator()** method.

68. **Explain the concept of a wildcard in Java generics.**

Answer:

- Wildcards in Java generics are used to represent unknown types in generic declarations.
- They allow flexibility in using generic types by enabling the declaration of methods or classes that can operate on different types.
- Wildcards are denoted by the **?** symbol and can be bounded (**? extends T** or **? super T**) to restrict the types allowed.

69. **What is the purpose of the java.util.concurrent package in Java?**

Answer:

- The **java.util.concurrent** package in Java provides utilities and classes for concurrent programming.
- It contains classes for creating and managing threads, thread pools, synchronization primitives, concurrent data structures, and utilities for handling concurrent execution.

70. **Describe the purpose of the java.util.stream package in Java.**

Answer:

- The **java.util.stream** package in Java provides a stream-based approach for processing collections of data.
- It introduces the concept of streams, which allow for declarative and parallel processing of data using functional-style operations like map, filter, reduce, and collect.
- Streams facilitate concise and expressive code for data processing tasks.