1. What is Java? Explain its features.

Answer:
Java is a high-level, object-oriented, platform-independent programming language developed by Sun Microsystems.

Key features:

- **Platform Independent:** Java code compiles to bytecode, which runs on JVM regardless of the platform.
- **Object-Oriented:** Supports classes, objects, inheritance, encapsulation, polymorphism, and abstraction.
- **Simple & Secure:** Syntax is simple, and Java has built-in security features like bytecode verification and sandbox environment.
- **Robust:** Provides strong memory management with automatic garbage collection and exception handling.
- **Multithreaded:** Supports concurrent execution of threads for efficient CPU utilization.

---

2. What are the main concepts of OOPS in Java?

Answer:
OOPS stands for Object-Oriented Programming System, which includes:

- **Encapsulation:** Wrapping data and methods into a single unit and restricting access via access modifiers.
- **Inheritance:** Child class inherits properties and behaviors of the parent class.
- **Polymorphism:** Ability to take multiple forms; method overloading (compile-time) and overriding (runtime).
- **Abstraction:** Hiding internal details and showing only functionality through abstract classes and interfaces.

---

3. What is the difference between JDK, JRE, and JVM?

Answer:

- **JDK (Java Development Kit):** Complete package for Java development; includes compiler (javac), JRE, and tools.
- **JRE (Java Runtime Environment):** Provides libraries and JVM to run Java applications but no compiler.
- **JVM (Java Virtual Machine):** Executes Java bytecode, provides platform independence by interpreting compiled code.

---

4. Explain the Java memory model (Heap, Stack).

Answer:

- **Heap:** Runtime area where all objects are stored. Garbage collector manages heap memory.
- **Stack:** Stores method calls, local variables, and references. Each thread has its own stack.

---

5. What is a constructor? What types of constructors are there?

Answer:
A constructor initializes a new object and has the same name as the class. It has no return type.

Types:

- **Default constructor:** No parameters, provided by JVM if none is defined.
- **Parameterized constructor:** Takes arguments to initialize the object.

---

6. What is method overloading and overriding?

Answer:

- **Overloading:** Same method name, different parameters (compile-time polymorphism).
- **Overriding:** Subclass provides a specific implementation of a superclass method (runtime polymorphism).

---

7. What is the difference between == and .equals() method?

Answer:

- == compares object references (memory location).
- .equals() compares the content of objects (can be overridden).

---

8. What are exceptions? What is the difference between checked and unchecked exceptions?

Answer:
Exceptions are runtime errors that disrupt normal program flow.

- **Checked exceptions:** Must be handled or declared (e.g., IOException).
- **Unchecked exceptions:** Runtime exceptions that don't need to be declared (e.g., NullPointerException).

---

9. What is multithreading? How is it useful?

Answer:
Multithreading is the concurrent execution of two or more threads for maximum CPU utilization. Useful for improving performance and responsiveness.

---

10. What is the difference between Array and ArrayList?

Answer:

- **Array:** Fixed size, stores primitives or objects.
- **ArrayList:** Resizable, part of Collections framework, stores objects and provides methods for manipulation.

11. What is the difference between String, StringBuilder, and StringBuffer?

Answer:

- String is immutable — once created, its value cannot change.
- StringBuilder is mutable and **not synchronized** — faster but not thread-safe.

- StringBuffer is mutable and **synchronized** — thread-safe but slower.

---

12. What are Java access modifiers?

Answer:
They control visibility of classes, methods, and variables:

- private: accessible only within the class.
- default (no modifier): accessible within the package.
- protected: accessible within package and subclasses.
- public: accessible everywhere.

---

13. What is an interface? How is it different from an abstract class?

Answer:

- An interface defines methods that a class must implement, with no method bodies (until Java 8).
- Abstract class can have method implementations and instance variables.
- A class can implement multiple interfaces but can inherit only one abstract class.

---

14. What is a Java package and what are its advantages?

Answer:
A package is a namespace grouping related classes and interfaces. It helps avoid name conflicts and controls access.

---

15. How does Java achieve platform independence?

Answer:
Java code compiles into bytecode, which runs on JVM. JVM abstracts away the underlying OS and hardware differences.

---

16. What is the role of the main() method in Java?

Answer:
main() is the entry point of any standalone Java application. JVM calls it to start execution.

---

17. What is an enum in Java?

Answer:
enum defines a fixed set of constants.

Example:

java

CopyEdit

enum Day { MON, TUE, WED }

Day today = Day.MON;

---

18. What is garbage collection in Java?

Answer:
Automatic process of reclaiming memory by removing objects no longer in use.

---

19. What are the main features of Java 8?

Answer:

- Lambda expressions
- Stream API
- Default and static methods in interfaces
- Optional class for null handling

---

20. What is a lambda expression?

Answer:
A concise way to represent anonymous functions, mainly used to implement functional interfaces.

Example:

java

CopyEdit

```
Runnable r = () -> System.out.println("Run in thread");
new Thread(r).start();
```

21. What is the difference between throw and throws?

Answer:

- throw is used to explicitly throw an exception from a method or block.
- throws declares exceptions that a method might throw to its caller.

---

22. What is the difference between ArrayList and LinkedList?

Answer:

- ArrayList uses a dynamic array, good for fast random access but slow for insertions/removals in the middle.
- LinkedList uses a doubly-linked list, good for fast insertions/removals but slower random access.

---

23. What is the purpose of the final keyword?

Answer:
final can be used to declare constants, prevent method overriding, and prevent inheritance.

---

24. What is a Java Bean?

Answer:
A Java Bean is a reusable software component that follows certain conventions:

- Has a no-arg constructor.
- Properties accessible via getters/setters.
- Implements Serializable.

---

25. What are the different ways to create a thread in Java?

Answer:

- Extend the Thread class and override run() method.
- Implement Runnable interface and pass it to a Thread object.

---

26. What is the difference between sleep() and wait() methods?

Answer:

- sleep() pauses the thread for a specific time without releasing the lock.
- wait() causes the thread to wait until notified and releases the lock.

---

27. What is a try-catch block?

Answer:
Used to handle exceptions. Code that might throw an exception goes in try, and catch handles the exception.

---

28. What is the use of the super keyword?

Answer:
Used to refer to the parent class instance, invoke parent class constructor or methods.

---

29. What is the difference between == and .equals() method?

Answer:

- == compares references.
- .equals() compares object content (if overridden).

---

30. What is the significance of the static keyword?

Answer:
Used for variables and methods belonging to the class rather than instances.

1. What are the main principles of Object-Oriented Programming (OOPS)?

Answer:
The main principles are:

- **Encapsulation:** Wrapping data and methods into a single unit and controlling access.
- **Inheritance:** Mechanism where one class acquires properties of another.
- **Polymorphism:** Ability to take many forms; includes method overloading and overriding.
- **Abstraction:** Hiding complex implementation details and showing only essential features.

---

2. What is encapsulation? How is it achieved in Java?

Answer:
Encapsulation restricts direct access to some of an object's components, which helps protect data integrity. It's achieved using **private** variables and **public getters and setters**.

---

3. What is inheritance? What are its types in Java?

Answer:
Inheritance allows a new class (child) to inherit properties and behavior from an existing class (parent). Types:

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
  (Java does not support multiple inheritance with classes, but supports it via interfaces)

---

4. What is polymorphism? Explain method overloading and overriding.

Answer:
Polymorphism means “many forms.”

- **Method overloading:** Same method name, different parameters (compile-time).
- **Method overriding:** Subclass provides specific implementation of a superclass method (runtime).

---

5. What is abstraction? How is it achieved in Java?

Answer:
Abstraction hides internal implementation and shows only the necessary features.
Achieved using **abstract classes** and **interfaces**.

---

6. What is the difference between an interface and an abstract class?

Answer:

Feature	Interface	Abstract Class
Method body	Only default/static methods (Java 8+)	Can have concrete and abstract methods
Multiple inheritance	Yes	No
Variables	public static final by default	Can have instance variables

---

Constructor Questions

---

7. What is a constructor in Java?

Answer:
A constructor is a special method used to initialize objects. It has the same name as the class and no return type.

---

8. What are the types of constructors in Java?

Answer:

- **Default constructor:** Provided by JVM if none defined; no arguments.
- **Parameterized constructor:** User-defined with parameters for initialization.
- **Copy constructor:** Not built-in in Java, but can be created manually to copy objects.

---

9. Can a constructor be private? If yes, why?

Answer:
Yes, to restrict object creation from outside the class. Used in Singleton design pattern.

---

10. What is constructor overloading?

Answer:
Having multiple constructors with different parameter lists in the same class.

---

11. What is the difference between a constructor and a method?

Answer:

Feature	Constructor	Method
Name	Same as class name	Any valid identifier
Return type	No return type	Has return type or void
Purpose	Initializes object	Performs specific task
Called	Automatically when object created	Called explicitly

Method Overloading Questions

---

1. What is method overloading in Java?

Answer:
Method overloading allows multiple methods in the same class to have the same name but different parameters (different type, number, or order).

---

2. What are the rules for method overloading?

Answer:

- Methods must have the same name.
- Parameter lists must differ in type, number, or order.
- Return type alone is not sufficient for overloading.

---

3. Can method overloading be achieved by changing only the return type?

Answer:
No. Changing only the return type without changing parameters causes a compilation error.

---

4. Does method overloading work with different access modifiers?

Answer:
Yes, methods can be overloaded regardless of access modifiers.

Method Overriding Questions

---

5. What is method overriding?

Answer:
Method overriding means redefining a method in the subclass that is already defined in the superclass with the same signature and return type.

---

6. What are the rules for method overriding?

Answer:

- Method must have the same name, parameter list, and return type (or covariant return type).
- Access modifier in subclass should be same or more accessible.
- Cannot override final or static methods.

---

7. What is the difference between method overloading and overriding?

Feature	Method Overloading	Method Overriding
Occurs in	Same class	Parent and child class
Method name	Same	Same
Parameters	Different	Same
Return type	Can be different	Same or covariant
Access modifier	Any	Same or more accessible
Compile/Runtime	Compile-time polymorphism	Runtime polymorphism

---

Access Modifiers Questions

---

8. What are access modifiers in Java?

Answer:
Access modifiers control visibility of classes, methods, and variables:

- private: accessible only within the class.
- default (no modifier): accessible within the package.
- protected: accessible within the package and subclasses.
- public: accessible from anywhere.

---

9. Can a subclass override a method with a more restrictive access modifier?

Answer:
No. The overriding method cannot have a more restrictive access modifier than the overridden method.

---

10. What is the default access modifier?

Answer:
If no access modifier is specified, it is **default (package-private)**, accessible within the same package only.

Abstract Classes Questions

---

1. What is an abstract class in Java?

Answer:
An abstract class is a class that cannot be instantiated and may contain abstract methods (without body) as well as concrete methods.

---

2. When should you use an abstract class?

Answer:
Use abstract class when you want to share common code among closely related classes but also enforce implementation of certain methods.

---

3. Can an abstract class have constructors?

Answer:
Yes, abstract classes can have constructors which are called when a subclass object is created.

---

4. Can an abstract class extend another class or implement interfaces?

Answer:
Yes, abstract classes can extend other classes and implement interfaces.

Interfaces Questions

---

5. What is an interface in Java?

Answer:
An interface is a reference type in Java, similar to a class, that can contain only abstract methods (until Java 7), default and static methods (Java 8+), and constants.

---

6. How is an interface different from an abstract class?

Feature	Interface	Abstract Class
Methods	Only abstract (until Java 7), default/static (Java 8+)	Abstract and concrete
Multiple inheritance	Supported	Not supported (only one class)
Variables	public static final	Instance variables allowed

---

7. Can a class implement multiple interfaces?

Answer:
Yes, Java supports multiple inheritance through interfaces.

---

8. What is a functional interface?

Answer:
An interface with exactly one abstract method, used for lambda expressions.

Collection Framework Questions

---

9. What is the Java Collection Framework?

Answer:
A set of interfaces and classes that implement commonly reusable collection data structures like List, Set, Queue, and Map.

---

10. What are the main interfaces in the Collection Framework?

Answer:

- **Collection** (root interface)
- **List** (ordered collection, allows duplicates)
- **Set** (no duplicates)
- **Queue** (FIFO data structure)
- **Map** (key-value pairs, not a subtype of Collection)

---

11. What is the difference between List, Set, and Map?

Interface	Characteristics
List	Ordered, allows duplicates
Set	Unordered, no duplicates
Map	Key-value pairs, keys unique

---

12. What is the difference between ArrayList and LinkedList?

Answer:

- ArrayList uses a dynamic array and provides fast random access.
- LinkedList uses a doubly linked list and provides faster insertions/removals.

---

13. What is the difference between HashMap and TreeMap?

Answer:

- HashMap stores entries with no order (hash table based).
- TreeMap stores entries sorted by keys (red-black tree based).

---

14. What is Iterator and how is it different from ListIterator?

Answer:

- Iterator can traverse collection in one direction and allows removal.
- ListIterator works only with List, supports bi-directional traversal and element modification.

1. Abstract Class Example

java

CopyEdit

```
abstract class Animal {
  abstract void sound();  // abstract method

  void sleep() {        // concrete method
    System.out.println("Sleeping...");
  }
}

class Dog extends Animal {
  void sound() {
    System.out.println("Bark");
  }
}
```

```
public class TestAbstract {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.sound();   // Output: Bark
        d.sleep();   // Output: Sleeping...
    }
}
```

---

2. Interface Example

java

CopyEdit

```
interface Drawable {
    void draw();  // abstract method
}

class Circle implements Drawable {
    public void draw() {
        System.out.println("Drawing Circle");
    }
}

public class TestInterface {
    public static void main(String[] args) {
        Drawable d = new Circle();
        d.draw();  // Output: Drawing Circle
    }
```

```
}
```

---

3. Functional Interface & Lambda Expression

```
java
CopyEdit
@FunctionalInterface
interface Calculator {
   int add(int a, int b);
}

public class TestLambda {
   public static void main(String[] args) {
      Calculator calc = (x, y) -> x + y;
      System.out.println(calc.add(5, 3));  // Output: 8
   }
}
```

---

4. Collection Framework: ArrayList Example

```
java
CopyEdit
import java.util.ArrayList;

public class TestArrayList {
   public static void main(String[] args) {
      ArrayList<String> fruits = new ArrayList<>();
      fruits.add("Apple");
```

```
        fruits.add("Banana");
        fruits.add("Orange");

        for (String fruit : fruits) {
            System.out.println(fruit);
        }
    }
}
```

---

5. Collection Framework: HashMap Example

java

CopyEdit

```
import java.util.HashMap;

public class TestHashMap {
    public static void main(String[] args) {
        HashMap<Integer, String> map = new HashMap<>();
        map.put(1, "One");
        map.put(2, "Two");
        map.put(3, "Three");

        for (Integer key : map.keySet()) {
            System.out.println(key + " : " + map.get(key));
        }
    }
}
```

---

6. Iterator Example

java

CopyEdit

```
import java.util.ArrayList;
import java.util.Iterator;

public class TestIterator {
   public static void main(String[] args) {
      ArrayList<String> list = new ArrayList<>();
      list.add("A");
      list.add("B");
      list.add("C");

      Iterator<String> it = list.iterator();
      while (it.hasNext()) {
         System.out.println(it.next());
      }
   }
}
```

---

1. Difference between Array and ArrayList

Feature	Array	ArrayList
Size	Fixed size	Dynamic size

Feature	Array	ArrayList
Data type	Can hold primitives and objects	Holds only objects (wrapper classes for primitives)
Performance	Faster access	Slightly slower due to dynamic resizing
Methods	Limited	Rich API (add, remove, etc.)

---

2. Difference between HashMap and Hashtable

Feature	HashMap	Hashtable
Synchronization	Not synchronized	Synchronized (thread-safe)
Null keys/values	Allows one null key and multiple null values	Does not allow null keys or values
Performance	Faster	Slower due to synchronization

---

3. Difference between == and .equals()

Feature	==	.equals()
Purpose	Compares object references	Compares object content
Usage	Works with primitives and objects	Typically overridden for content equality
Default behavior	Reference equality	Reference equality (unless overridden)

---

4. Difference between abstract class and interface

Feature	Abstract Class	Interface
Methods	Can have abstract and concrete methods	Only abstract methods (Java 7 and earlier), default/static allowed in Java 8+

Feature	Abstract Class	Interface
Multiple inheritance	No	Yes
Variables	Instance variables allowed	Only public static final variables

---

5. Difference between StringBuilder and StringBuffer

Feature	StringBuilder	StringBuffer
Synchronization	Not synchronized	Synchronized (thread-safe)
Performance	Faster	Slower
Usage	Single-threaded environment	Multi-threaded environment

---

6. Difference between Checked and Unchecked exceptions

Feature	Checked Exception	Unchecked Exception
Checked at compile time?	Yes	No
Example	IOException	NullPointerException
Handling	Must be handled or declared	Optional to handle

---

7. Difference between final, finally, and finalize

Keyword	Purpose
final	Used to declare constants, methods or classes which cannot be overridden/extended
finally	Block that executes after try-catch, used for cleanup
finalize	Method called by garbage collector before object destruction