**Abstraction**

Abstraction is a principle of hiding implementation details and showing only essential information to the user. In Java, abstraction is achieved through abstract classes and interfaces.

java

Copy code

abstract class Animal {

abstract void sound();

}

class Dog extends Animal {

void sound() {

System.out.println("Woof");

}

}

**2. Interface**

An interface in Java is a reference type that contains only constants, method signatures, default methods, static methods, and nested types.

java

Copy code

interface Animal {

void sound();

}

class Dog implements Animal {

public void sound() {

System.out.println("Woof");

}

}

**3. Abstract Class**

An abstract class is a class that cannot be instantiated on its own and can have abstract and non-abstract methods.

java

Copy code

abstract class Vehicle {

abstract void start();

void fuelUp() {

System.out.println("Fueling...");

}

}

**4. Abstract Class vs Interface**

| **Abstract Class** | **Interface** |
| --- | --- |
| Can have both abstract and concrete methods | Only abstract methods (in older Java versions) |
| Can have fields | Cannot have fields (prior to Java 8) |
| Supports multiple inheritance indirectly | Supports multiple inheritance directly |
| Can have constructors | Cannot have constructors |

**5. Marker Interface**

A marker interface is an empty interface with no fields or methods, used to indicate that a class belongs to a particular type. Example: Serializable.

**6. Encapsulation**

Encapsulation is the bundling of data and methods that operate on that data into a single unit, typically a class, while restricting access to some of the object's components using access modifiers.

java

Copy code

class Employee {

private int age;

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

**7. Type Casting**

Type casting is the process of converting a variable from one type to another.

**8. Upcasting and Downcasting**

* **Upcasting** is casting a subclass object to a superclass type.
* **Downcasting** is casting a superclass reference back to a subclass type.

java

Copy code

Animal a = new Dog(); // Upcasting

Dog d = (Dog) a; // Downcasting

**9. Boxing, Unboxing, and AutoBoxing**

* **Boxing**: Converting a primitive data type into its corresponding wrapper class.
* **Unboxing**: Converting a wrapper class back to its primitive type.
* **AutoBoxing**: The automatic conversion between primitive types and their wrapper classes.

java

Copy code

int x = 10;

Integer boxed = x; // AutoBoxing

int unboxed = boxed; // Unboxing

**10. Implicit and Explicit Typecasting**

* **Implicit**: Automatic type conversion by the compiler.
* **Explicit**: Manually casting one type to another.

java

Copy code

double d = 10; // Implicit casting

int i = (int) d; // Explicit casting

**11. Equality vs equals Method**

* == checks if two references point to the same object.
* equals() checks if two objects are logically equivalent.

java

Copy code

String a = new String("test");

String b = new String("test");

System.out.println(a == b); // false

System.out.println(a.equals(b)); // true

**12. Object Class**

The Object class is the parent class of all classes in Java. It provides methods like toString(), equals(), hashCode(), clone(), etc.

**13. toString Method**

The toString() method returns a string representation of the object.

java

Copy code

public String toString() {

return "Employee: " + this.name;

}

**14. equals Method**

The equals() method compares the content of objects for equality.

java

Copy code

public boolean equals(Object obj) {

if (this == obj) return true;

if (obj == null || getClass() != obj.getClass()) return false;

Employee employee = (Employee) obj;

return this.name.equals(employee.name);

}

**15. hashCode Method**

The hashCode() method returns an integer value that is used in hashing-based collections like HashMap.

java

Copy code

public int hashCode() {

return Objects.hash(name, id);

}

**16. clone Method**

The clone() method creates a copy of an object.

java

Copy code

public Object clone() throws CloneNotSupportedException {

return super.clone();

}

**17. getClass Method**

The getClass() method returns the runtime class of an object.

java

Copy code

Class<?> objClass = this.getClass();

**18. wait, notify, notifyAll**

These methods are used in multithreading to handle the communication between threads.

* wait(): Causes the current thread to wait until another thread invokes notify().
* notify(): Wakes up a single thread that is waiting on the object's monitor.
* notifyAll(): Wakes up all threads that are waiting on the object's monitor.

java

Copy code

synchronized(obj) {

obj.wait();

obj.notify();

}

**19. finalize Method**

The finalize() method is invoked by the garbage collector before the object is destroyed.

java

Copy code

protected void finalize() throws Throwable {

System.out.println("Object is being garbage collected.");

}