**1. Covariant Type Method Overriding**

In Java, method overriding allows a subclass to provide a specific implementation of a method that is already defined by its superclass. Covariant return types allow the overridden method to return a subtype of the return type of the method in the superclass.

java

Copy code

class Animal {

Animal getAnimal() {

return new Animal();

}

}

class Dog extends Animal {

@Override

Dog getAnimal() { // Covariant return type (Dog is a subclass of Animal)

return new Dog();

}

}

**2. Method Hiding**

In Java, static methods can be hidden, but not overridden. When a subclass defines a static method with the same signature as a static method in the superclass, it hides the superclass's method.

java

Copy code

class Parent {

static void display() {

System.out.println("Parent static method");

}

}

class Child extends Parent {

static void display() { // Method hiding

System.out.println("Child static method");

}

}

public class Test {

public static void main(String[] args) {

Parent.display(); // Calls Parent's static method

Child.display(); // Calls Child's static method

}

}

**3. New Keyword**

The new keyword is used to create new instances (objects) of a class.

java

Copy code

class Dog {

String name;

Dog(String name) {

this.name = name;

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog("Buddy"); // Using new keyword

System.out.println(dog.name);

}

}

**4. Constructor**

A constructor is a special method used to initialize objects. It is called when an object of a class is created.

**5. Default Constructor**

A default constructor is automatically provided by Java if no constructors are explicitly defined.

java

Copy code

class Animal {

// Default constructor is implicitly provided

}

**6. Non-Parameterized Constructor**

A constructor without parameters.

java

Copy code

class Car {

Car() {

System.out.println("Car created");

}

}

public class Main {

public static void main(String[] args) {

Car car = new Car(); // Calls non-parameterized constructor

}

}

**7. Parameterized Constructor**

A constructor that takes arguments to initialize an object.

java

Copy code

class Car {

String model;

Car(String model) {

this.model = model;

}

}

public class Main {

public static void main(String[] args) {

Car car = new Car("Tesla"); // Calls parameterized constructor

System.out.println(car.model);

}

}

**8. Types of Constructors**

* **Default constructor**: No parameters.
* **Parameterized constructor**: Accepts parameters.

**9. Instance Block**

Instance blocks are used to initialize instance variables and are executed every time an object is created.

java

Copy code

class Example {

{

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

}

Example() {

System.out.println("Constructor executed");

}

}

public class Main {

public static void main(String[] args) {

new Example(); // Both instance block and constructor will execute

}

}

**10. Constructor Chaining (Same Class)**

One constructor calls another constructor in the same class.

java

Copy code

class Example {

Example() {

this(10); // Calling parameterized constructor

System.out.println("Default constructor");

}

Example(int x) {

System.out.println("Parameterized constructor: " + x);

}

}

public class Main {

public static void main(String[] args) {

new Example(); // Constructor chaining within the same class

}

}

**11. Constructor Chaining (Parent Class)**

A subclass calls a constructor of its parent class using super().

java

Copy code

class Parent {

Parent() {

System.out.println("Parent constructor");

}

}

class Child extends Parent {

Child() {

super(); // Calls Parent constructor

System.out.println("Child constructor");

}

}

public class Main {

public static void main(String[] args) {

new Child(); // Constructor chaining with parent class

}

}

**12. Constructor Never Becomes Part of Inheritance**

Constructors are not inherited by subclasses.

**13. Anonymous Object**

An object that is instantiated but not assigned to a reference variable.

java

Copy code

class Dog {

void bark() {

System.out.println("Dog barks");

}

}

public class Main {

public static void main(String[] args) {

new Dog().bark(); // Anonymous object

}

}

**14. Static Block**

A block that is executed when the class is loaded, even before object creation.

java

Copy code

class Example {

static {

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

}

}

public class Main {

public static void main(String[] args) {

new Example(); // Static block is executed before this

}

}

**15. Final Keyword**

The final keyword can be used with classes, methods, and variables to restrict modification.

java

Copy code

class Example {

final int value = 10; // Final variable cannot be modified

final void display() { // Final method cannot be overridden

System.out.println("Final method");

}

}

**16. Call by Value and Call by Reference**

Java is **call by value**, meaning it passes a copy of the value to methods.

java

Copy code

class Example {

void modifyValue(int x) {

x = 10; // Doesn't affect the original value

}

}

public class Main {

public static void main(String[] args) {

int a = 5;

new Example().modifyValue(a);

System.out.println(a); // Output: 5 (Call by value)

}

}

**17. Packages, Default Package, Import, and Package Keyword**

A **package** is used to group related classes.

java

Copy code

// Example of creating a package

package myPackage;

class Example {

public void display() {

System.out.println("Inside myPackage");

}

}

// Importing a package

import myPackage.Example;

public class Main {

public static void main(String[] args) {

Example ex = new Example();

ex.display();

}

}

**18. Static Import**

Allows static members (fields and methods) of a class to be used without qualifying them with the class name.

java

Copy code

import static java.lang.Math.\*;

public class Main {

public static void main(String[] args) {

System.out.println(sqrt(16)); // No need to write Math.sqrt

}

}