1. class Base {

public Base() {

System.**out**.println("Base");

}

}

class Derived extends Base {

public Derived() {

System.**out**.println("Derived");

}

}

class DeriDerived extends Derived {

public DeriDerived() {

System.**out**.println("DeriDerived");

}

}

public class Test {

public static void main(String[] args) {

Derived b = new DeriDerived();

}

}

a)Base

Derived

DeriDerived

b)Derived

DeriDerived

c)DeriDerived

Derived

Base

d)DeriDerived

Derived

**Answer:**

**aBase**

**Derived**

**DeriDerived**

**Explanation:**

Whenever a class gets instantiated, the constructor of its base classes (the constructor of the root of the hierarchy gets executed first) gets invoked before the constructor of the instantiated class.

**2.**

abstract class Car {

static {

System.**out**.print("1");

}

public Car(String name) {

super();

System.**out**.print("2");

}

{

System.**out**.print("3");

}

}

public class BlueCar extends Car {

{

System.**out**.print("4");

}

public BlueCar() {

super("blue");

System.**out**.print("5");

}

public static void main(String[] gears) {

new BlueCar();

}

}

a) 23451

b) 12354

c) 13245

d) The code does not compile.

**Answer:**

**c) 13245**

**Explanation:**

The class is loaded first, with the static initialization block called and 1 is outputted first. When the BlueCar is created in the main() method, the superclass initialization happens first. The instance initialization blocks are executed before the constructor, so 32 is outputted next.

Finally, the class is loaded with the instance initialization blocks again being called before the constructor, outputting 45. The result is that 13245 is printed, making Option C the correct answer.

**3**. class Parent{

public void className(){

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

}

}

class Child extends Parent{

void className(){

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

}

}

public class Test{

public static void main(String[] args){

Parent parent = new Child();

parent.className();

}

}

a) Parent

b) Child

c) Compile-time error

d) Run-time error

**Answer:**

**c) Compile-time error**

**Explanation:**

When overriding a parent class method in a child class, we cannot reduce the visibility of the method. For example, if the method is defined as public in the parent class, a child class cannot override it with protected. The code will give the compilation error “Cannot reduce the visibility of the inherited method from Parent”.

**4.** class Demo{

public Demo(int i){

System.**out**.println("int");

}

public void Demo(short s){

System.**out**.println("short");

}

}

public class Test{

public static void main(String[] args){

short s = 10;

Demo demo = new Demo(s);

}

}

a) int

b) short

c) Compile-time error

d) Run-time error

**Answer:**

1. **int**

**Explanation**:

The class Demo has one constructor i.e. with int argument. The short value is automatically promoted to an int value during object creation so the constructor with the int argument will be called and it will print “int”.

**5.** class Automobile {

private String drive() {

return "Driving vehicle";

}

}

class Car extends Automobile {

protected String drive() {

return "Driving car";

}

}

public class ElectricCar extends Car {

@Override

public final String drive() {

return "Driving an electric car";

}

public static void main(String[] wheels) {

final Car car = new ElectricCar();

System.**out**.print(car.drive());

}

}

A. Driving vehicle

B. Driving an electric car

C. Driving car

D. The code does not compile

**Answer :**

**B. Driving an electric car**

**Explanation:**

The drive() method in the Car class does not override the version in the Automobile class since the method is not visible to the Car class.

The drive() method in the ElectricCar class is a valid override of the method in the Car class, with the access modifier expanding in the subclass. For these reasons, the code compiles, and Option D is incorrect.

In the main() method, the object created is an ElectricCar, even if it is assigned to a Car reference. Due to polymorphism, the method from the ElectricCar will be invoked, making Option B the correct answer.

**6.**

public class MyClass {

private int value;

public MyClass() {

value = 10;

}

public MyClass(int val) {

this();

value += val;

}

public int getValue() {

return value;

}

public static void main(String[] args) {

System.out.println(new MyClass(5).getValue());

}

}

a) 5

b) 10

c) 15

d) 20

**Answer: c) 15**

**Explanation:** The constructor MyClass(int val) calls this() to invoke the default constructor, setting value to 10. Then, it adds the parameter val. So, value becomes 15.

**7.**

public class NumberHolder {

private int x;

private int y;

public NumberHolder(int x, int y) {

this.x = x;

this.y = y;

}

public NumberHolder(double z) {

this((int) z, (int) z);

}

public int sum() {

return x + y;

}

public static void main(String[] args) {

System.**out**.println(new NumberHolder(3.7).sum());

}

}

public class NumberHolder {

private int x;

private int y;

public NumberHolder(int x, int y) {

this.x = x;

this.y = y;

}

public NumberHolder(double z) {

this((int) z, (int) z);

}

public int sum() {

return x + y;

}

public static void main(String[] args) {

System.**out**.println(new NumberHolder(3.7).sum());

}

}

a) 3

b) 6

c) 7

d) 8

**Answer: a) 3**

**Explanation:** The constructor NumberHolder(double z) converts z to an integer and calls the primary constructor. So, x and y both become 3.

**8.** class Shape {

void draw() {

System.**out**.println("Drawing a shape");

}

}

class Circle extends Shape {

void draw() {

System.**out**.println("Drawing a circle");

}

}

public class PolymorphismQuestion1 {

public static void main(String[] args) {

Shape shape = new Circle();

shape.draw();

}

}

What will be the output of the above code?

a) Drawing a shape

b) Drawing a circle

c) Compilation error

d) Runtime error

**Answer: b) Drawing a circle**

**Explanation**: The draw method of the Circle class will be invoked at runtime due to dynamic polymorphism. Even though the reference variable is of type Shape, the actual object is of type Circle.

**9.**class A {

void display() {

System.**out**.println("Class A");

}

}

class B extends A {

void display() {

System.**out**.println("Class B");

}

}

public class PolymorphismQuestion3 {

public static void main(String[] args) {

A obj1 = new B();

B obj2 = new B();

obj1.display();

obj2.display();

}

}

What will be the output of the above code?

a) Class A, Class A

b) Class B, Class B

c) Class B, Class A

d) Compilation error

**Answer: c) Class B, Class A**

**Explanation:** For obj1, the reference type is A, but the object is of type B, so the display method of B will be invoked. For obj2, the display method of B will be called directly.

**10.**public class StringQuestion1 {

public static void main(String[] args) {

String str1 = "Hello";

String str2 = str1.concat(" World");

System.**out**.println(str1);

System.**out**.println(str2);

}

}

What will be the output of the above code?

a) Hello, Hello World

b) Hello, World

c) Hello World, Hello World

d) Hello World, World

**Answer: a) Hello, Hello World**

**Explanation**: Strings in Java are immutable. The concat method creates a new string with the concatenated value. The original string (str1) remains unchanged.

**11.**public class StringQuestion3 {

public static void main(String[] args) {

String str1 = "Hello";

String str2 = str1.toUpperCase();

System.**out**.println(str1);

System.**out**.println(str2);

}

}

What will be the output of the above code?

a) Hello, Hello

b) Hello, HELLO

c) hello, HELLO

d) HELLO, HELLO

**Answer: a) Hello, Hello**

**Explanation:** The toUpperCase method creates a new string with all characters converted to uppercase. The original string (str1) remains unchanged.

**12.**interface Animal {

void makeSound();

}

class Dog implements Animal {

public void makeSound() {

System.**out**.println("Woof");

}

}

public class InterfaceQuestion2 {

public static void main(String[] args) {

Animal myPet = new Dog();

myPet.makeSound();

}

}

a) Meow  
b) Woof  
c) Compilation error  
d) Runtime error

**Explanation:** The **makeSound** method of the **Dog** class will be invoked at runtime due to interface implementation. The actual object is of type **Dog**.

**Answer: b) Woof**

**13.**

interface Printable {

void print();

}

class Printer implements Printable {

public void print() {

System.**out**.println("Printing...");

}

}

public class InterfaceQuestion3 {

public static void main(String[] args) {

Printable p = new Printer();

((Printer) p).print();

}

}

a) Printing...  
b) Compilation error  
c) Runtime error  
d) No output

**Explanation:** The **print** method of the **Printer** class will be invoked at runtime. The cast is necessary to access the specific methods of the implementing class.

**Answer: a) Printing...**