Polymorphism

1. Find Output

```
class A
{
   protected void getData()
   {
      System.out.print("A");
   }
}
class B extends A
{
   protected void getData()
   {
      System.out.print("B");
   }
}

public class Test
{
   public static void main(String[] args)
   {
      A a = new B();
      a.getData();
   }
}
```

Output

a) B
b) A
c) AB
d) BA

Ans: a

2. Find Output

```
class A
{
   public void getData()
   {
      System.out.print("A");
   }
}
class B extends A
{
   protected void getData()
   {
      System.out.print("B");
   }
}

public class Test
{
   public static void main(String[] args)
   {
      A a = new B();
      a.getData();
   }
}
```

Output

a) Compilation error
b) Runtime error
c) A
d) B

Ans: a

3. Find Output

```
class A
{
protected void getData()
   {
      System.out.print("A");
   }
}
class B extends A
{
   void getData()
   {
      System.out.print("B");
   }
}

public class Test
{
   public static void main(String[] args)
   {
      A a = new B();
      a.getData();
   }
}
```

Output

a) Compilation error
b) Runtime error
c) A
d) B

Ans: a

4. Find Output

```
class A
{
public void getDataOne()
   {
      System.out.print("A");
   }
}
class B extends A
{
public void getData()
   {
      System.out.print("B");
   }
}

public class Test
{
   public static void main(String[] args)
   {
      A a = new B();
      a.getData();
   }
}
```

Output

a) Compilation error
b) Runtime error
c) A
d) B

Ans: a

5.Find Output

```
class A
{
public void getDataOne()
   {
      System.out.print("A");
   }
}
class B extends A
{
public void getData()
   {
      System.out.print("B");
   }
}

public class Test
{
   public static void main(String[] args)
   {
      B a = new B();
      a.getData();
   }
}
```

Output

a) B
b) BA
c) AB
d) A

Ans: a

6. Find Output

```
class Test
{
   void myMethod()
   {
      System.out.print("A");
   }
}
public class Derived extends Test
{
   void myMethod()
   {
      System.out.print("B");
   }

   public static void main(String[] args)
   {
      Derived object = new Test();
      object.myMethod();
   }
}
```

Output

- a) A
  b) B
  c) Compilation error
  d) Runtime error
- **Ans.** (c)
  Explanation: A child class cannot be used as a reference to an object of super class.

7. Find Output

```
class A
{
public void getDataOne()
   {
      System.out.print("A");
   }
}
class B extends A
{
public void getData()
   {
      System.out.print("B");
   }
}

public class Test
{
   public static void main(String[] args)
   {
      Object a = new A();
      a.getDataOne();
   }
}
```

Output

a) Compilation error
b) Runtime error
c) A
d) B

Ans: a

8. Find Output

```
class Derived
{
   protected final void getDetails()
   {
      System.out.println("Derived class");
   }
}

public class Test extends Derived
{
   protected final void getDetails()
   {
      System.out.println("Test class");
   }
   public static void main(String[] args)
   {
      Derived obj = new Derived();
      obj.getDetails();
   }
}
```

Output

a) Derived class
b) Test class
c) Runtime error
d) Compilation error

Ans. (d)
Explanation: Final and static methods cannot be overridden.

9. Find Output

```
class Derived
{
   public void getDetails(String temp)
   {
      System.out.println("Derived class " + temp);
   }
}

public class Test extends Derived
{
   public int getDetails(String temp)
   {
      System.out.println("Test class " + temp);
      return 0;
   }
   public static void main(String[] args)
   {
      Test obj = new Test();
      obj.getDetails("GFG");
   }
}
```

Output

- a) Derived class GFG
  b) Test class GFG
  c) Compilation error
  d) Runtime error
- **Ans.** (c)
  Explanation: The overriding method must have same signature, which includes, the argument list and the return type.

10. Find Output

```
class Derived
{
   public void getDetails()
   {
      System.out.println("Derived class");
   }
}

public class Test extends Derived
{
   protected void getDetails()
   {
      System.out.println("Test class");
   }
   public static void main(String[] args)
   {
      Derived obj = new Test();  // line xyz
      obj.getDetails();
   }
}
```

Output

- a) Test class
  b) Compilation error due to line xyz
  c) Derived class
  d) Compilation error due to access modifier
- **Ans:** (d)
  Explanation: The overriding method can not have more restrictive access modifier.

11. Find Output

```
public class Test
{
   public int getData() //getdata() 1
   {
      return 0;
   }
   public long getData() //getdata 2
   {
      return 1;
   }
   public static void main(String[] args)
   {
      Test obj = new Test();
      System.out.println(obj.getData());
   }
}
```

Output

a) 1
b) 0
c) Runtime error
d) Compilation error

Ans. (d)
Explanation: For method overloading, **methods must have different signatures**. Return type of methods does not contribute towards different method signature, so the code above give compilation error. Both getdata 1 and getdata 2 only differ in return types and NOT signatures.

12. Find Output

```
public class Test
{
   public int getData(String temp) throws IOException
   {
      return 0;
   }
   public int getData(String temp) throws Exception
   {
      return 1;
   }
   public static void main(String[] args)
   {
      Test obj = new Test();
      System.out.println(obj.getData("GFG"));
   }
}
```

Output

- a) 0
  b) 1
  c) Compilation error
  d) Runtime error
- **Ans.** (c)
  Explanation: Methods that throws different exceptions are not overloaded as their signature are still the same.

13. Find Output

```
public class Test
{
   private String function()
   {
      return ("A");
   }
   public final static String function(int data)
   {
      return ("B");
   }
   public static void main(String[] args)
   {
      Test obj = new Test();
      System.out.println(obj.function());
   }
}
```

Output

a) A
b) B
c) Compilation error
d) Runtime error

Ans: a

Explanation: Access modifiers associated with methods doesn’t determine the criteria for overloading. The overloaded methods could also be declared as final or static without affecting the overloading criteria.

14. Find Output

```
public class Test
{
   private String function(String temp, int data)
   {
      return ("A");
   }
   private String function(int data, String temp)
   {
      return ("B");
   }
   public static void main(String[] args)
   {
      Test obj = new Test();
      System.out.println(obj.function(4, "C"));
   }
}
```

Output

a) B
b) A
c) BA
d) Compilation error

Ans: a

15. Find Output

```
public class Test
{
   private String function(String temp, int data, int sum)
   {
      return ("Raju");
   }
   private String function(String temp, int data)
   {
      return ("Rani");
   }
   public static void main(String[] args)
   {
      Test obj = new Test();
      System.out.println(obj.function("Babu", 0, 20));
   }
}
```

Output

a) Raju
b) Rani
c) Babu
d) Compilation error

Ans: a

16. Find Output

```
public class Test
{
   private String function(float i, int f)
   {
      return ("Raju");
   }
   private String function(double i, double f)
   {
      return ("Rani");
   }
   public static void main(String[] args)
   {
      Test obj = new Test();
      System.out.println(obj.function(1. , 20));
   }
}
```

Output

a) Rani
b) Raju
c) Compilation error
d) Runtime error

Ans: a

Explanation: This one is really simple. Different type of arguments contribute towards method overloading as the signature of methods is changed with type of attributes. Whichever matches the attributes is obviously called in Method Overloading. Here we are pass first attributes double not float.

17. Find Output

Which polymorphism behavior do you see in below class?

```
class Paint {
        public void Color(int x) {
        }

        public void Color(int x, int y) {
        }

        public void Color(int x, int y, int z) {
        }
}
```

Output

a) Method overloading
b) Constructor overloading
c) Method overriding
d) Run time polymorphism

Answer: a

Method with same name with different number of arguments or same number of arguments with different data types is method overloading in java programming.

18. Find Output

```
class Father {

		public void car() {
			System.out.println("Father's Car");
		}
}

class Son extends Father {

		public void car() {
			System.out.println("Son's Car");
		}
}

public class Sample {

		public static void main(String[] args) {

			Son john = new Son();
			john.car();
		}

}
```

Output

a) Father’s Car

b) Son’s Car

c) There is an ambiguity, so no one Car

d) Compiler Error

Answer: B

Find Output

Output