Moderate Marks 1 m each

Which of the following is NOT a pillar of object-oriented programming?
a. Inheritance
b. Abstraction
c. Polymorphism
d. Concurrency

Which design pattern is used to create objects without specifying the exact class to create?
a. Abstract Factory
b. Singleton
c. Prototype
d. Factory Method

Which of the following is NOT a valid access modifier in Java?
a. public
b. protected
c. private
d. internal

Which design pattern is used to ensure that only one instance of a class is created?
a. Singleton
b. Factory Method
c. Observer
d. Decorator

Which of the following is an example of polymorphism?
a. Overloading a method
b. Overriding a method
c. Both A and B
d. Neither A nor B

Easy 1m each

What is inheritance in Java?
a. A way to create objects of a class
b. A way to reuse code from an existing class
c. A way to modify the behavior of a method
d. A way to declare variables in a class

What is a constructor in Java?
a. A method that returns a value
b. A method that takes no arguments
c. A method that creates an object of a class
d. A method that modifies an existing object

What is the purpose of the "static" keyword in Java?
a. To make a method non-static
b. To make a variable non-static
c. To create a constant variable
d. To associate a method or variable with the class, rather than with instances of the class

What is the difference between an interface and a class in Java?
a. A class can have abstract methods, while an interface cannot
b. A class can have a constructor, while an interface cannot
c. A class can be instantiated, while an interface cannot
d. All of the above

What is polymorphism in Java?
a. The ability of a method to have multiple implementations
b. The ability of a class to inherit from multiple superclasses
c. The ability of a class to have multiple constructors
d. The ability of a method to call itself recursively

Hard 2 marks each

Match each term on the left with the appropriate description on the right:

A	Default constructor	1	A constructor that creates a new object as a copy of an existing object
B	Parameterized constructor	2	A constructor that is only accessible within the class itself
C	Copy constructor	3	A constructor with no parameters
D	Private constructor	4	A constructor that takes one or more parameters

Options:

A-3 B-1 C-4 D-2
A-3 B-4 C-2 D-1
A-3 B-4 C-1 D-2
A-1 B-3 C-4 D-2

What is the output of the following code?

```
class Animal {
  public void makeSound() {
    System.out.println("Animal is making a sound");
  }
}
class Dog extends Animal {
  public void makeSound() {
    System.out.println("Bark!");
  }
}
public class Main {
  public static void main(String[] args) {
    Animal a = new Dog();
    a.makeSound();
  }
}
```

a. "Animal is making a sound"
b. "Bark!"
c. An error will occur
d. no output

What is the output of the following code?

```
class Animal {
  public Animal() {
    System.out.println("Animal constructor");
  }
}
class Dog extends Animal {
  public Dog() {
    System.out.println("Dog constructor");
  }
}
public class Main {
  public static void main(String[] args) {
    Dog d = new Dog();
  }
}
```

a. "Animal constructor" and "Dog constructor"
b. "Dog constructor" and "Animal constructor"
c. "Animal constructor"
d. "Dog constructor"

What is the output of the following code?

```
class Animal {
  public void makeSound() {
    System.out.println("Animal is making a sound");
  }
}
class Dog extends Animal {
  public void makeSound() {
    super.makeSound();
    System.out.println("Bark!");
  }
}
public class Main {
  public static void main(String[] args) {
    Dog d = new Dog();
    d.makeSound();
  }
}
```

a. "Animal is making a sound" and "Bark!"
b. "Bark!" and "Animal is making a sound"
c. "Animal is making a sound"
d. "Bark!"

What will be the output of the code when run?

```
public class Animal {
  protected int legs;
  public Animal(int legs) {
    this.legs = legs;
  }
}
public class Dog extends Animal {
  public Dog(int legs) {
    super(legs);
  }
  public void printLegs() {
    System.out.println("I have " + legs + " legs.");
  }
}
public class Main {
  public static void main(String[] args) {
    Dog d1 = new Dog(4);
    Animal a1 = d1;
    a1.legs = 3;
    d1.printLegs();
  }
}
```

a. "I have 3 legs."
b. "I have 4 legs."
c. Compilation error because the Dog class cannot be cast to the Animal class.
d. Compilation error because the Animal class is abstract and cannot be instantiated.