Inheritance, constructors and access modifiers

Agenda

Key terms

Constructor

> a constructor (abbreviation: ctor) is a special type of subroutine called to create an object. It prepares the new object for use, often accepting arguments that the constructor uses to set required member variables.

Access modifier

> An access modifier defines the accessibility of a class, attribute, method or constructor. There are four types of access modifiers in Java: public, protected, default (no modifier), and private.

Inheritance

> Inheritance is a mechanism wherein a new class is derived from an existing class. In Java, classes may inherit or acquire the properties and methods of other classes. A class derived from another class is called a subclass, whereas the class from which a subclass is derived is called a superclass.

Constructors

A constructor is a special method that is called when an object is created. It is used to initialize the object. It is called automatically when the object is created. It can be used to set initial values for object attributes.

Constructors are the gatekeepers of object-oriented design. Let us create a class for students:

```
public class Student {

    private String name;
    private String email;
    private Integer age;
    private String address;
    private String batchName;
    private Integer psp;

    public void changeBatch(String batchName) {
        ...
    }
}
```

The above class can be used to create objects of type `Student`. This is done by using the `new` keyword:

```
Student student = new Student();
student.name = "Eklavya";
```

You can notice that we did not define a constructor for the `Student` class. This brings us to our first type of constructor

Default constructor

A default constructor is a constructor created by the compiler if we do not define any constructor(s) for a class.

A default constructor is a constructor that either has no parameters, or if it has parameters, all the parameters have default values. If no user-defined constructor exists for a class and one is needed, the compiler implicitly declares a default parameterless constructor.

A default constructor is also known as a no-argument constructor or a nullary constructor. All fields are left at their initial value of 0 (integer types), 0.0 (floating-point types), false (boolean type), or null (reference types) An example of a no-argument constructor is:

```
public class Student {
    private String name;
    private String email;
    private Integer age;
    private String address;
    private String batchName;
    private Integer psp;

    public Student() {
        // no-argument constructor
    }
}
```

Notice a few things about the constructor which we just wrote. First, it's a method, but it has no return type. That's because a constructor implicitly returns the type of the object that it creates. Calling new `Student()` now will call the constructor above. Secondly, it takes no arguments. This particular kind of constructor is called a no-argument constructor.

Syntax of a constructor

In Java, every class must have a constructor. Its structure looks similar to a method, but it has different purposes. A constructor has the following format `<Constructor Modifiers> <Constructor Declarator> <Constructor Body>`

Constructor declarations begin with access modifiers: They can be public, private, protected, or package access, based on other access modifiers. Unlike methods, a constructor can't be abstract, static, final, native, or synchronized.

The declarator is the name of the class, followed by a parameter list. The parameter list is a comma-separated list of parameters enclosed in parentheses. The body is a block of code that defines the constructor's behavior.

```
Constructor Name (Parameter List)
```

Parameterised constructor

Now, a real benefit of constructors is that they help us maintain encapsulation when injecting state into the object. The constructor above is a no-argument constructor and hence value have to be set after the instance is created.

```
Student student = new Student();
student.name = "Eklavya";
student.email = "ek@drona.in";
```

The above approach works but requires setting the values of all the fields after the instance is created. Also, we won't be able to validate or sanitize the values. We can add the validation and sanitization logic in the getters and setters but we wont be able to fail instance creation. Hence, we need to add a parameterised constructor. A parameterised constructor has the same syntax as the constructors before, the onl change is that it has a parameter list.

```
public class Student {
    private String name;
    private String email;

    public Student(String name, String email) {
        this.name = name;
        this.email = email;
    }
}
```

Now the objects can be created with the following syntax:

```
Student student = new Student("Eklavya", "ek@drona.in");
```

In Java, constructors differ from other methods in that:

- Constructors never have an explicit return type.
- Constructors cannot be directly invoked (the keyword "new" invokes them).
- Constructors should not have non-access modifiers.

Access modifiers

There are two types of modifiers in Java: access modifiers and non-access modifiers.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of access modifiers in Java:

- `public` - The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package
- `protected` - The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
- `private` - The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- `default` - The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

Inheritance

Inheritance is the mechanism that allows one class to acquire all the properties from another class by inheriting the class. We call the inheriting class a child class and the inherited class as the superclass or parent class.

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Imagine, as a car manufacturer, you offer multiple car models to your customers. Even though different car models might offer different features like a sunroof or bulletproof windows, they would all include common components and features, like engine and wheels.

It makes sense to create a basic design and extend it to create their specialized versions, rather than designing each car model separately, from scratch.

In a similar manner, with inheritance, we can create a class with basic features and behavior and create its specialized versions, by creating classes, that inherit this base class. In the same way, interfaces can extend existing interfaces.

Let us create a new class `User` which should be the parent class of `Student`:

```
public class User {
    private String name;
    private String email;

    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }
}
```

Now, let us create a new class `Student` which should be the child class of `User`. Let us add some methods specific to the student class:

```
public class Student {
    private String batchName;
    private Integer psp;

    ...
}
```

Now in order to inherit the methods and fields of the parent class, we need to use the keyword `extends`:

```
public class Student extends User {
    private String batchName;
    private Integer psp;

    ...
}
```

To pass the values to the parent class, we need to create a constructor and use the keyword `super`:

```
public class Student extends User {
    private String batchName;
    private Integer psp;

    public Student(String name, String email, String batchName, Integer
psp) {
        super(name, email);
        this.batchName = batchName;
        this.psp = psp;
    }
}
```

Types of inheritance

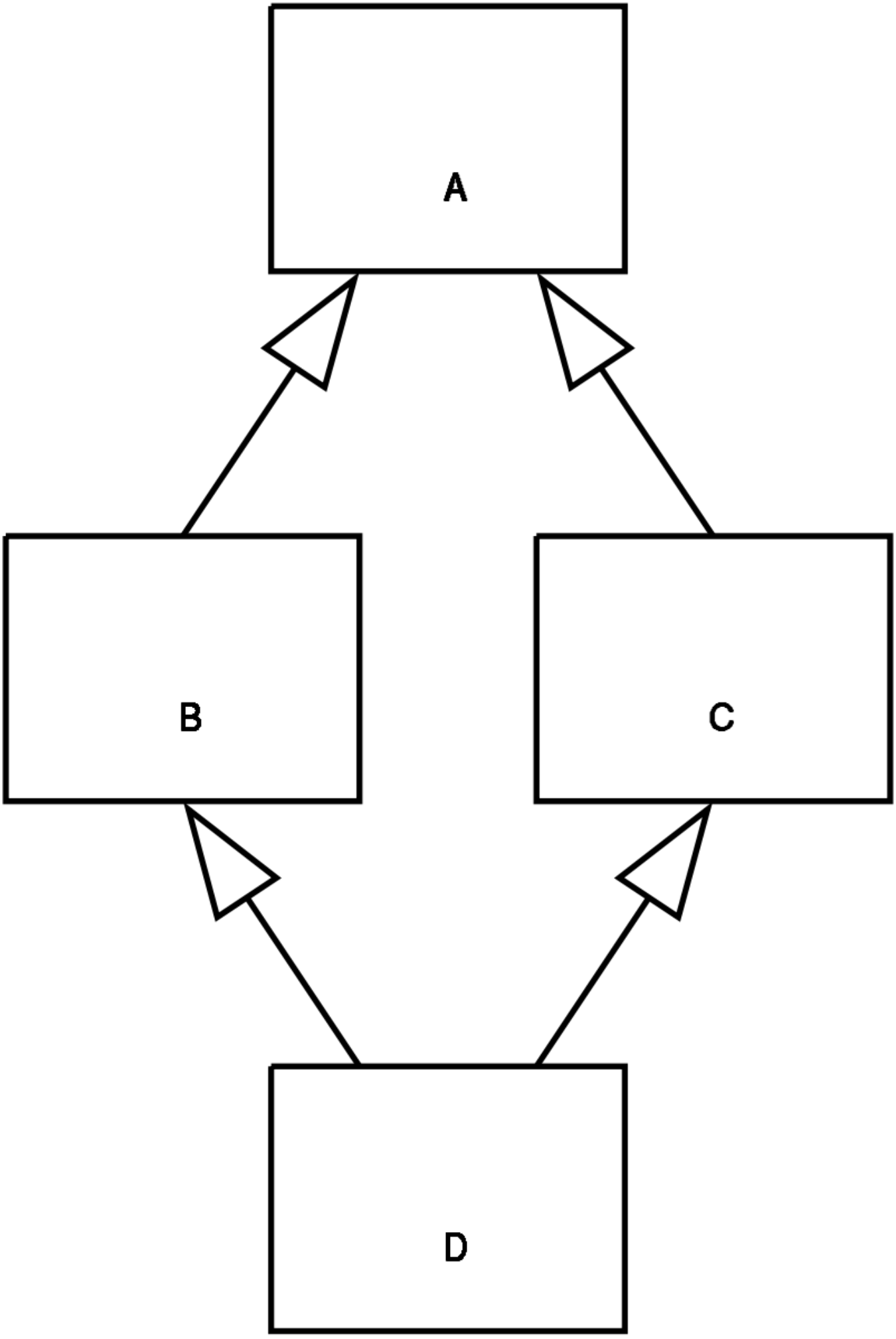
There are four types of inheritance:

- `Single` - A single inheritance is when a class can have only one parent class.
- `Multilevel` - A multilevel inheritance is when a class can have multiple parent classes at different levels.
- `Hierarchical` - When two or more classes inherits a single class, it is known as hierarchical inheritance.
- `Multiple` - When a class can have multiple parent classes, it is known as multiple inheritance.

 Inheritance

Diamond problem

In multiple inheritance one class inherits the properties of multiple classes. In other words, in multiple inheritance we can have one child class and n number of parent classes. Java does not support multiple inheritance (with classes).


A
B
C
D

The "diamond problem" (sometimes referred to as the "Deadly Diamond of Death") is an ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C. If there is a method in A that B and C have overridden, and D does not override it, then which version of the method does D inherit: that of B, or that of C.

Reading list

- Deadly Diamond of Death
- Detailed explanation of the diamond problem
- Sealed classes