Access modifiers, constructors and the static keyword

Key terms

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.

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 modifiers

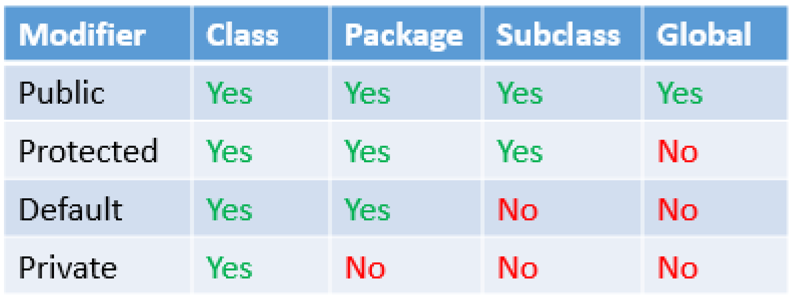
There are two types of modifiers in Java: access modifiers and non-access modifiers.

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

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:

```java
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:

```java
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.

Copy constructor

A copy constructor is a member function that initializes an object using another object of the same class. A copy constructor has the following general function prototype:

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

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

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

The static keyword

The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than instance of the class.

Static methods are methods that can be called without creating an instance of the class. They are declared using the static keyword.

Why use a static method?

- You can call a static method without creating an instance of the class.
- You don't need to have separate implementations of the same method for each instance of the class.

How to create a static method?

Let us create a static method for a Person

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

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

    public static String getPersonInfo(Person person) {
        return person.getName() + " " + person.getEmail();
    }
}
```

Now let's create a class that uses the static method:

```
public class User {
    public static void main(String[] args) {
        Person person = new Person("John", "Doe");

        System.out.println(Person.getPersonInfo(person));
    }
}
```

Reading list