Inheritance in Java

Mentoring Meeting - 8/21/2024

Baha K.

What is inheritance?

- Inheritance allows one class to inherit content from another class (fields and methods).
    - Allows code **reusability.**

- **Base class / Super class / Parent class**
    - The class where the fields and methods are inherited from.

- **Derived class / Sub class / Child class**
    - The class which inherits the properties of parent class.

'extend' keyword

- Used to indicate that a new class is derived from the base class.

```
public class Dog extends Animal {

}
```

  - Dog is the child class and Animal is the parent class.

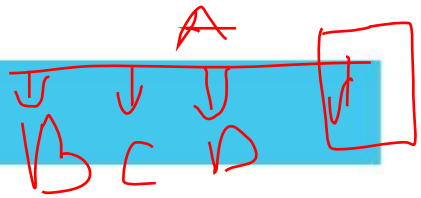
What is inherited?

- All public variables and methods.
- All protected variables and methods.
- All default variables and methods if the super and subclasses are in the same package.

What is not inherited?

- Constructors are not inherited.
- Any private variables or methods are not inherited.

Types of inheritance



Single inheritance

```
public class A{

}

public class B extends A{

}
```

Multi-level inheritance

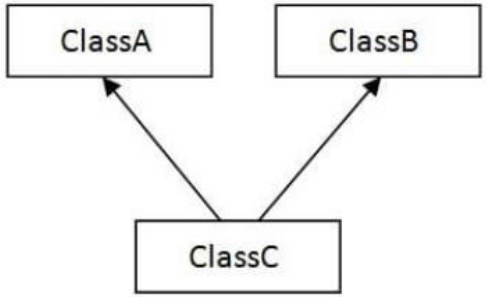
```
public class A{

}

public class B extends A{

}

public class C extends B{

}
```

Hierarchical inheritance

```
public class A{

}

public class B extends A{

}

public class C extends A{

}

public class D extends A{

}
```

Multiple inheritance

- Java **DOES NOT** support multiple inheritance with classes.
  - One class cannot have more than one superclass and inherit features from all parent classes.



- Multiple inheritance can be achieved through **interfaces**, because a class can implement multiple interfaces.

What is ‘is-a’ relationship?

- The ‘**is-a**’ relationship in Java refers to the inheritance relationship between classes, where a subclass **‘is-a’** type of its superclass.

```
public class Animal {

    }
public class Dog extends Animal {

    }
public class GermanShepherd extends Dog {

    }
```

- German Shepherd is a dog.
- Dog is an animal.
- German Shepherd is an animal.

Static members in inheritance

- Static members from a super class are inherited **as long as the access modifier allows it.**
- Static variables are shared class variables, it will have a **single central value for all objects** and subclasses.
- Static methods, can be called either using ParentClass.methodName or SubClass.methodName.

What is method overriding?

- Method overriding allows a child class to provide a **specific implementation** of a method that is already provided by the parent class.
- Why do we need method overriding?
    - Sometimes the method inherited from its superclass is not enough for the subclass's purpose. Because the **subclass is more specialized** than the superclass, so it is sometimes necessary for the subclass to replace inadequate superclass methods with more suitable ones.

Method Overriding Rules

1. There must be **is-a relationship** (inheritance) → **No inheritance, no overriding.**
2. The method must have **the same name** as in the parent class.
3. The method must have **the same parameter** as in the parent class.
4. Access modifier: Needs to be **the same or more visible**
   - public - > public
   - protected - > protected, public
   - default - > default, protected, public
5. Return type:
   - must be **the same** or
   - **covariant type** (same class type or subclass type) (if the return type is object)

Covariant return type, Java'da bir metot geçersiz kılarken üst sınıfta belirtilen dönüş tipinin, geçersiz kılınan metotta aynı sınıfın alt sınıfı olarak belirtilebilmesi anlamına gelir.

java — Kodu kopyala

```java
class Animal {
    // Üst sınıfta bir metot tanımı
    public Animal getAnimal() {
        return new Animal();
    }
}

class Dog extends Animal {
    // Alt sınıfta metot geçersiz kılma (overriding) ve covariant return type kullanımı
    @Override
    public Dog getAnimal() {
        return new Dog();
    }
}
```

@Override annotation

- Indicates that the child class method is over-writing its base class method.
- **Not** mandatory.
- What is the purpose of @Override?
    - **Provides compile-time checks** to ensure that the overridden method is actually defined in the superclass.
    - If there is a typo or a mistake in the method name or signature, the compiler will generate an error indicating that the method is not actually overriding a method in the superclass.

Compile-time checks, Java'da hataların kod çalıştırılmadan önce, yani derleme sırasında tespit edilmesini sağlayan mekanizmalardır. Bu, tür uyumsuzlukları, erişim hataları, sözdizimi hataları ve diğer mantıksal hataları içerir.

Can we override static methods?

- A static method **cannot** be overridden in Java. If you declare another static method with same signature in child class than the static method of parent class will be hidden, and any call to that static method in child class will go to static method declared in that class itself.

- This is known as **method hiding** in Java.

Why static methods cannot be overridden?

- Static methods **cannot be overridden** in the same way as instance methods. When a static method is defined in a superclass, it belongs to the class itself rather than any particular instance of the class. Therefore, when a subclass defines a static method with the same name as the one in the superclass, it is simply **hiding** the superclass's static method, **not overriding** it.

- The method overriding is based upon **dynamic binding at runtime** (decisions about which method to use is made while the program is running) and static methods are bonded using **static binding at compile time.** This means static methods are resolved even before objects are created, that's why it's not possible to override static methods in Java.

- When you call a static method on a subclass, it will execute the static method defined in that subclass, regardless of whether a method with the same name exists in the superclass. The static method in the superclass will still be accessible through the superclass itself, but it will not be associated with the subclass in any way.

Method overloading vs method overriding

Method Overloading	Method Overriding
Occurs in the **same** class.	Occurs between **two or more classes** that have an IS-A relationship
Parameters must be **different**.	Parameters must be the **same**.
Access specifiers can be **different**.	Access specifiers **must not be more restrictive** than that of the original method.
private and final methods **can** be overloaded.	private and final methods **cannot** be overridden.
Return type of the method does not matter; it **can be the same or different.**	Return type **must be the same or covariant.**
Compile-time polymorphism	**Run-time** polymorphism

'super' keyword

- A reference variable that is used to **refer to parent class objects.**

It can be used to:

1. **Access parent class constructors:**
   - In a child class constructor, the super() keyword can be used to call the constructor of the superclass. This ensures that the parent class's initialization is performed before the child class's initialization.
2. **Access parent class methods:**
   - Helpful when the subclass overrides the method, but you still want to call the superclass's implementation. By using super.methodName(), you can explicitly invoke the method in the superclass.
3. **Access parent class fields:**
   - If a subclass has a field with the same name as a field in the superclass, the super keyword can be used to refer to the superclass's field. This is useful when you want to access or modify the superclass's field within the subclass.

How to run constructors in inheritance?

Three rules:

1. In an inheritance relationship, the superclass constructor always executes **before** the subclass constructor.

2. The **super()** keyword refers to an object's superclass. You can use the super() keyword to call a superclass constructor.

3. If a subclass constructor does not explicitly call a superclass constructor, Java will automatically call the superclass's default constructor, or no-arg constructor, just before the code in the subclass constructor is executed.

'super' keyword

super() :

- (super() is used to call the parent class constructor from the child class constructor.
- Parameters **MUST** match with the parent constructor.
- It needs to be the first statement in the child class constructor.
- this() also needs to be the first statement in the constructor, so super() and this() **CANNOT** be in the same constructor.
- If you do not add super() in your constructor, compiler will put one for you.
- If parent class only has constructor with parameters, then child constructor **MUST** make a matching super(params) call.

this() vs super()

Keyword	Usage	Purpose
this()	Inside a constructor	Refers to another constructor within the same class. It is used to call a different constructor in the same class, allowing for constructor chaining.
super()	Inside a constructor	Refers to a constructor in the parent class. It is used to call a constructor of the superclass, allowing for initialization of the parent class's state before the child class's state.

Final methods and variables

- All methods and variables can be overridden by default in sub classes.

- To prevent subclasses from overriding the members of the super class, declare them as final using ‘final’ keyword.
  - final int length = 20;
  - Final void showLength(...) {...}

- Defining method final ensures that functionality of defined method will not be altered.

- Similarly, the value of final variable will never be changed.

Object class

- Defined in the java.lang package of the Java standard class library.
- All classes are derived from the Object class.
- If a class is not explicitly defined to be the child of an existing class, it is assumed to be the child of the Object class.
- Therefore, the Object class is the ultimate root of all class hierarchies.

Object class

- The Object class contains a few useful methods, which are inherited by all classes.
- For example, toString() method is defined in the Object class.
- Every time we call toString(), we have actually been overriding an existing definition.
- All objects are guaranteed to have a toString() method via inheritance.
- Thus, the println method can call toString for any object that is passed to it.