FILKOM

Inheritance in Generalization Relationship

Object-oriented Programming

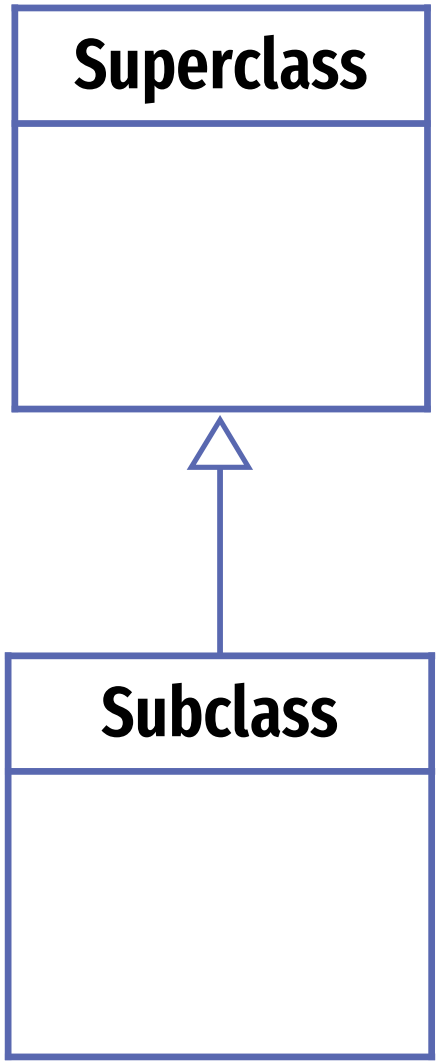
Aryo Pinandito, S.T., M.MT, Ph.D.

Review: Class Relationship

- Is-A, a generalization relationship representation when a class represent a more general form of another class
- Uses-A, a dependency relationship representation when a class uses an object of another class
- Has-A, an association relationship when an object of a class becomes a member of another class.
  - Aggregation and Composition

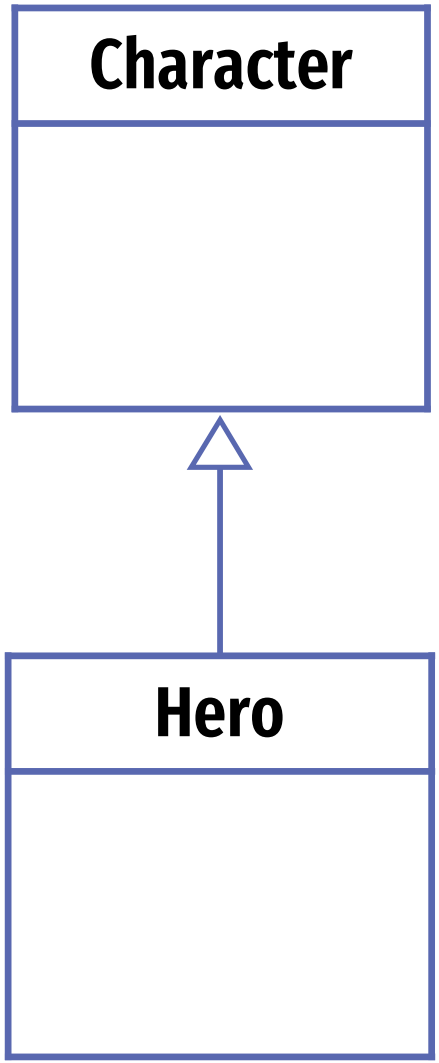
Generalization: "Is-A"

- Generalization establishes a relationship between a more general class (known as superclass) and a more specialized class (known as subclass).
- Is-A relationship defines the relationship between two classes in which one class extends another class



Generalization: "Is-A"

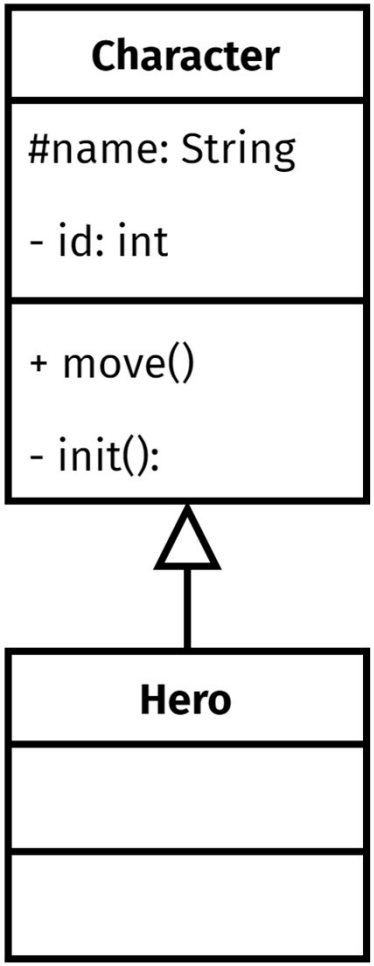
```
class Character {
  protected int hp;
}

class Hero extends Character {
  ...
}
```



Inheritance in Generalization

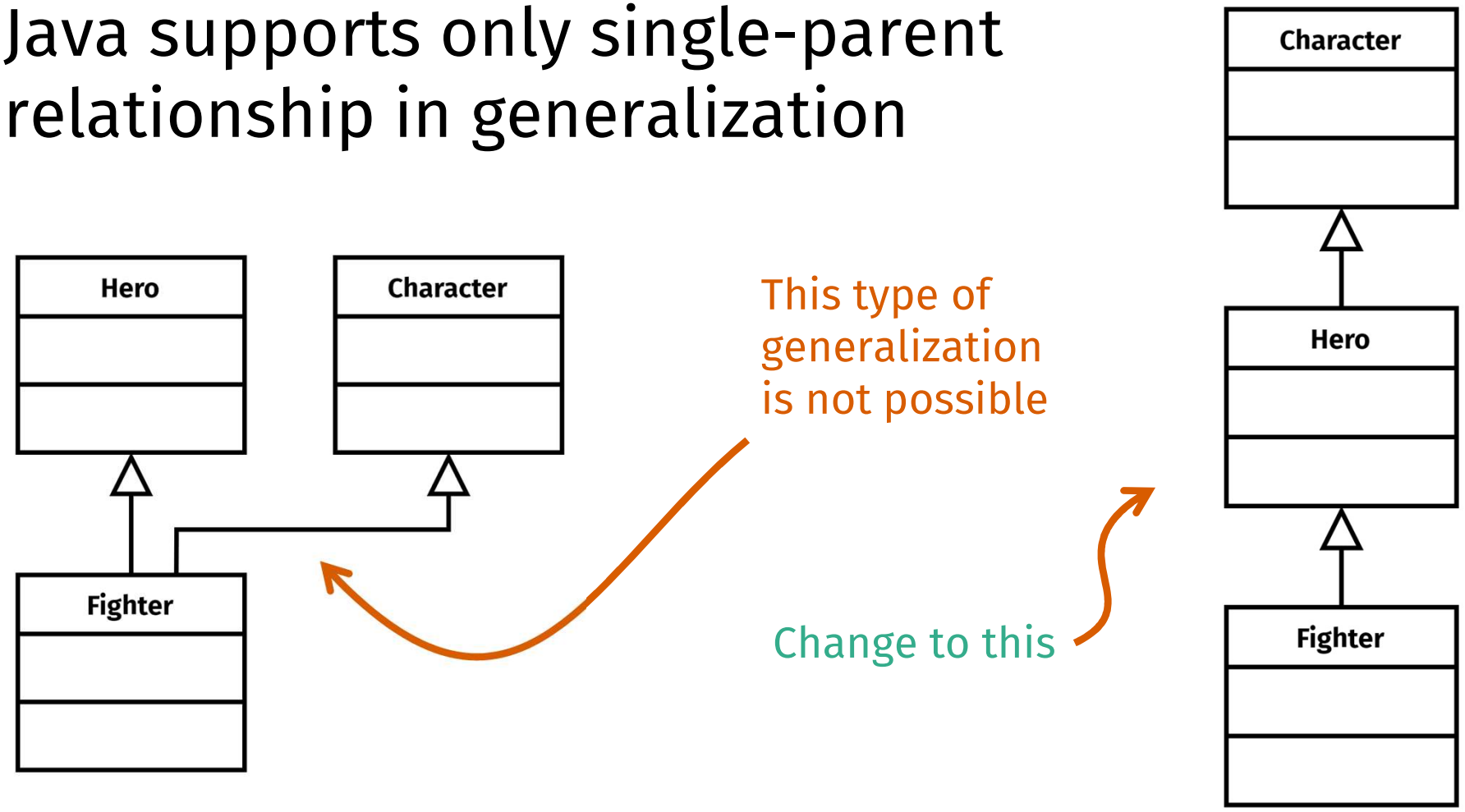
- Generalization relationship is also known as "inheritance" relationship
  - non-private members (attributes and methods) of superclass were inherited to the subclass.
- In generalization, a subclass inherits all the members of its superclass, except for those with private modifier.

```
public class Character {
  protected String name;
  private int id;
  public void move() { ... }
  private void init() { ... }
}

public class Hero extends Character {}

Hero hero = new Hero();
hero.move(); // inherited

hero.init();    // Error, not inherited
print(hero.id); // Error, not inherited
```



Multilevel Inheritance

- Java supports only single-parent relationship in generalization



The code below is OK

```
public class Character {}
public class Hero extends Character {}
public class Fighter extends Hero {}
```

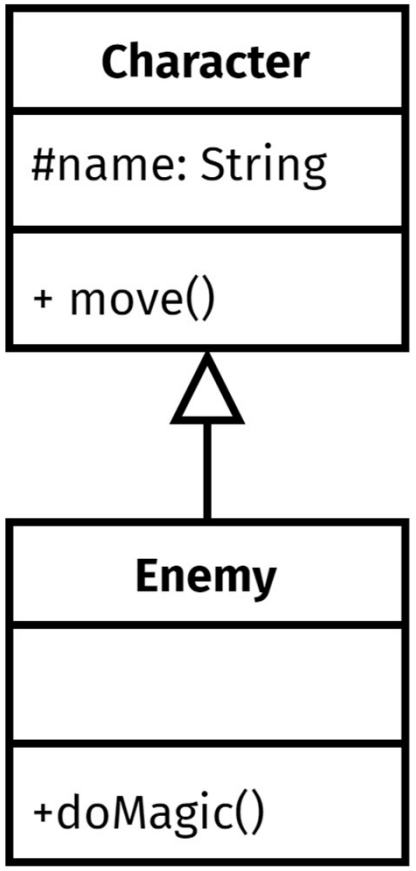
But this one is not OK

```
public class Character {}
public class Fighter extends Character, Hero {}
// Not allowed, a class cannot extends
// two or more classes at a time
```

Inheritance in Generalization

- The superclass-subclass relationship in generalization may also represents a parent-child relationship.
  - Child class may inherits its parents' members
- In generalization, subclass is more specific than its superclass
  - Subclass may have their own members (attributes and methods)

```
public class Character {
  protected String name;
  public void move() { ... }
}

public class Enemy extends Character {
  public void doMagic() { ... }
}

Enemy witch = new Enemy();
witch.move();    // inherited
witch.doMagic(); // Enemy's own method
```

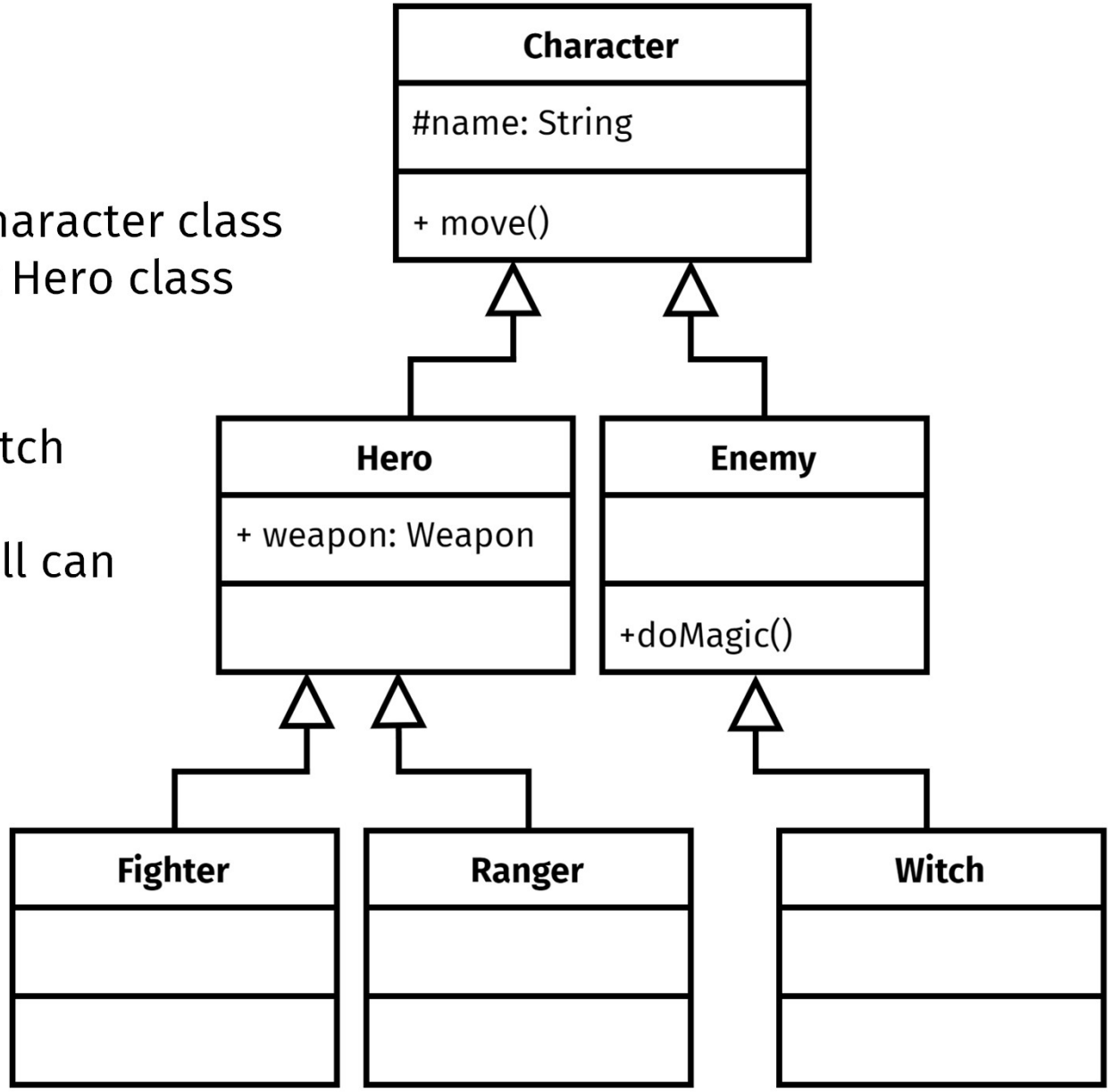


Inheritance Example

Hero and Enemy inherit Character class
Fighter and Ranger inherit Hero class
Witch inherit Enemy class

All Fighter, Ranger, and Witch have name inherited from Character class and they all can also move()

Only Witch can doMagic(), but Fighter and Ranger both own a weapon.



```
public class Character {
  protected String name;
  public void move() { ... }
}

public class Hero extends Character {
  public Weapon weapon;
}

public class Enemy extends Character {
  public void doMagic() { ... }
}
```

```
public class Fighter extends Hero {}
public class Ranger extends Hero {}
public class Witch extends Enemy {}

Fighter chronos = new Fighter();
Ranger drow = new Ranger();
Witch hermione = new Witch();

chronos.move(); drow.move(); juniper.move();
chronos.weapon = new Weapon("sword");
drow.weapon = new Weapon("bow");
hermione.doMagic();

// chronos.doMagic();
// hermione.weapon = new Weapon("staff");
```

Generates error; try to access to members that are not inherited.

java.lang.Object Class

- All Java classes inherit java.lang.Object class.
- If a class does not extends another class, it is implicitly extends the java.lang.Object class.

```
public class Hero {
  ...
}
```

is equals to:

```
public class Hero extends Object {
  ...
}
```

```
Hero fighter = new Hero();
Hero ranger = new Hero();

// access to methods below
// were inherited from the
// java.lang.Object class
String s = ranger.toString();
Boolean b = fighter.equals(ranger);
```

FILKOM

Method Override

Method Override

- A subclass may override (change/hide) the implementation of a superclass method.
- Overriding method of subclass must have the same signature with overridden method of superclass
  - same name, parameters type and orders
- Overriding method uses @Override annotation before its declaration

```
public class Character {
  public void say(String words) {
    print("Hello! " + words);
  }
}

public class Hero extends Character {
  @Override
  public void say(String words) {
    print("Yo! " + words);
  }
}
```

```
Character c = new Character();
c.say("Bruh..."); // Hello! Bruh...
Hero h =  new Hero();
h.say("Bruh..."); // Yo! Bruh...
```

Method Override

- A class method can be overridden during object instantiation.
- Replace a method implementation only for a particular object.

```
public class Character {
  public void say(String words) {
    print("Hello! " + words);
  }
}
Character chA = new Character();  // Instantiation
Character chB = new Character() { // Instantiation
  @Override // and Override
  public void say(String words) {
    print("Yo! " + words);
  }
};

chA.say("Bruh..."); // Hello! Bruh...
chB.say("Bruh..."); // Yo! Bruh...
```

Overloading Not Overriding

- When a method of subclass have the different signature with a method of superclass it is called method overload.
  - i.e., one of the parameter type or parameter orders is different
- Method overload does not change/hide method of superclass of the same name.
- Overloading method cannot have @Override annotation

```
public class Character {
  public void say() {
    print("Hello!");
  }
}

public class Hero extends Character {
  // no @Override
  public void say(String words) {
    print("Yo! " + words);
  }
}
```

```
Hero h =  new Hero();
h.say(); // Hello!
h.say("Bruh..."); // Yo! Bruh...
```

Overload vs Override

Overload

- Two or more method have the same name, but different parameter signature
  - different type, number, and/or order
- Can be implemented in a single class
- Can be implemented in two or more different class having generalization relationship

Override

- Two or more method have the same name and have the same parameter signature
  - same type, number, and order
- Can only be implemented in generalization relationship, abstract class, or implementation of interface

FILKOM

Static, Super, and Final Keyword in Generalization

Keyword super

- On a method override, subclass may call overridden method of superclass using the super keyword
  - Thus, reveals the hidden/overridden method of superclass

```
public class Character {
  public void say(String words) {
    print("Hello! " + words);
  }
}

public class Hero extends Character {
  @Override
  public void say(String words) {
    print("Yo! " + words);
    super.say(words);
  }
}
```

```
Hero h =  new Hero();
h.say("Bruh...");
// Yo! Bruh... Hello! Bruh...
```

Keyword super for Constructor

- In generalization relationship, superclass constructors are not inherited.
- Superclass constructor must be called from the extending class (subclass) using super() as the first statement of subclass constructor
- Superclass constructor with no parameter will be called implicitly even if it is not called from subclass constructor

```
public class Character {
  Character() {
    print("Hello character!");
  }
}

public class Hero extends Character {
  Hero() {
    print("Hello hero!");
  }
}

Hero h =  new Hero();
// Hello character! // constructor is called implicitly
// Hello hero!
```

```
public class Character {
  Character() {
    print("Hello character!");
  }
}

public class Hero extends Character {
  Hero() {
    super(); // call to superclass' as first statement!
    print("Hello hero!");
  }
}

Hero h =  new Hero();
// Hello character! // constructor is called explicitly
// Hello hero!
```

Keyword super for Constructor

- Using non-empty-parameter constructor of subclass will implicitly call superclass' empty-parameter constructor;
  - Unless, super() with non-empty-parameter is called explicitly

```
public class Character {
  Character() { // empty parameter
    print("Hello empty character!");
  }
  Character(String name) { // one String parameter
    print("Hello " + name + " character!");
  }
}

public class Hero extends Character {
  Hero() { // empty parameter
    print("Hello empty hero!");
  }
  Hero(String name) { // one String parameter
    print("Hello " + name + " hero!");
  }
}
```

```
Hero h = new Hero("Superman");
// Hello empty character!
// Hello Superman hero!


Hero(String name) {
  // explicitly call superclass' constructor
  // that requires one String parameter
  super(name);
  print("Hello " + name + " hero!");
}


Hero h = new Hero("Superman");
// Hello Superman character!
// Hello Superman hero!
```

Keyword static

- Static method cannot be overridden

```
public class Character {
  public static void say(String words) { ... }
}

public class Hero extends Character {
  @Override
  public void say(String words) { ... } // Error
}
```

Keyword static

- If a static method is redeclared in subclass, the static method of superclass will be hidden.

```
public class Character {
  public static void say(String words) {
    print("Hello! " + words);
  }
}

public class Hero extends Character {
  // no @Override
  public static void say(String words) {
    print("Yo! " + words);
  }
}

Hero.say("Bruh...");
// Yo! Bruh...
```

Keyword final

- Final method cannot be overridden

```
public class Character {
  public final void say(String words) { ... }
}

public class Hero extends Character {
  @Override
  public void say(String words) { ... } // Error
}
```

Keyword final

- Final class cannot be extended

```
public final class Character {
  public void say(String words) { ... }
}

public class Hero extends Character { // Error
  public void say(String words) { ... }
}
```

Questions?

Assignment

- Berdasarkan assignment sebelumnya, buatlah class Character yang merupakan bentuk general dari Hero dan Enemy
- Lakukan analisis atribut dan method terhadap class Hero dan Enemy
  - Pindahkan atribut dan method dari class Hero dan Enemy yang dapat digeneralisasikan ke dalam class Character
- Lakukan generalisasi pada class Hero dan Enemy terhadap class Character

Assignment

- Tambahkan atribut -name:String pada class Character dan buat method accessor pada class Character
- Tambahkan constructor pada Character untuk mengisi atribut name tersebut pada saat object Hero dan Enemy diinisialisasi
- Tambahkan constructor lain yang juga digunakan untuk menginisialisasi nilai-nilai atribut name, level, dan hp

Assignment

- Buat class Sword dan Bow yang merupakan bentuk spesialisasi dari class Weapon.
- Buat constructor yang menginisialisasi atribut weapon berikut:
  - atk: int
  - name: String
  - isBroken: boolean
  - condition: int
- Override method use() yang diwariskan pada class Bow

Assignment

- Buat kode program Java-nya, dan gambarkan diagram class-nya.