

JAVA / LESSON 21 OF 40

The Mob Family Tree

Inheritance, super and method rewriting

This chapter uses JDK 21 and does not require the preview function. | 90-110 minutes

Making an enemy family `EnemyFamily`

All three monsters are injured, but each has its own attack method. You will put the common rules into the parent class, let the subclasses only write their own differences, and then check whether there are any wrong branches in the family tree.

Prerequisites: classes, objects, construction methods and encapsulation. Today we are moving the duplicate biological codes into the same family tree.

Before you start: It is recommended to complete Lessons 17-20 first

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=21&lang=en>

Long code lines may wrap at the > margin marker. The marker is not part of the code.

Inheritance: The hollow triangle points to the parent class

Enemy

Subclass	Superclass	Relationship
Zombie	Enemy	is an enemy
Skeleton	Enemy	is an enemy
Creeper	Enemy	is an enemy

In this lesson

01. Before starting: predict first, then let the program answer
02. Inheritance first answers whether it is a
03. The parent class saves common parts
04. extends to the family tree
05. super completes parent class initialization
06. Override rewrite attack behavior
07. Three kinds of enemies and three kinds of attacks
08. super method reuses parent class behavior
09. Access rights guard the boundaries
10. How to choose inheritance and combination
11. Three types of test family rules
12. Challenge Blueprint Enemy Family
13. Challenge Realizing the Enemy Family
14. Expansion tasks and clearance review
15. Own acceptance record

01 / 15

Before starting: predict first, then let the program answer

This chapter uses JDK 21 and does not require the preview function. The story is borrowed from the block world, and the example runs on the ordinary Java console; if the values and rules are course models, they will be clearly stated.

Each common example is a complete independent file, saved according to the file name marked on the box. Place each example in a different folder to avoid duplicate names of auxiliary classes. When the complete project is divided into sections, they are connected sequentially in the same file.

Terminal command: replace the Chinese placeholder with the actual file name and class name

```
javac -encoding UTF-8 ExampleFile.java
java ExampleClass
```

- Prediction: Write the expected output first, or draw out the changes in references and data.
- Verification: Check line by line after running, and then only change one condition.
- Explanation: Explain whether the results come from Java rules, course contracts, or input data.

"Intentional error" is used to recognize bugs and cannot be mixed with ordinary examples. Use a special temporary directory for file experiments; do not use real archives to test coverage and deletion.

02 / 15

Inheritance first answers whether it is a

Zombies, skeletons, and creepers all have names, lives, and injury behaviors. Copy these common codes three times, and you have to chase three monsters when fixing bugs.

Inheritance allows subclasses to reuse the status and behavior of the parent class, but the premise is that the subclass is indeed a type of parent class.

Inheritance: The hollow triangle points to the parent class

Enemy

Subclass	Superclass	Relationship
Zombie	Enemy	is an enemy

Subclass	Superclass	Relationship
Skeleton	Enemy	is an enemy
Creeper	Enemy	is an enemy

Put the subclass name into the sentence "So-and-so is an Enemy." When a sentence is unnatural, you should usually use a field or combination instead.

Conditions for passing this lesson

- Understand the family tree of parent and child classes.
- Let the subclass constructor call super.
- Use `@Override` to override methods with the same name and parameters.
- Prepare normal, borderline, and error tests for three types of enemies.

03 / 15

The parent class saves common parts

The parent class is not a large glove box. Only members that are needed by all enemies and have the same meaning are suitable for inclusion in the Enemy.

Only one copy of common fields and methods is maintained

Element	Explanation
name	Only one copy of the name is saved
health	Private life field
takeDamage	Unified injury rules

The check that the life is not lower than 0 remains in the parent class. The three subclasses share the same rule and don't have to invent their own versions.

L21Example02.java · Complete independent example

```
class Enemy {
    protected final String name;
    private int health;
    Enemy(String name, int health) {
        this.name = name == null || name.isBlank()
            ? "Unnamed" : name.strip();
        this.health = Math.max(1, Math.min(40, health));
    }
    public void takeDamage(int amount) {
        if (amount > 0) health = Math.max(0, health - amount);
    }
    public int health() { return health; }
    public String attack() { return name + " Normal attack 1"; }
    public String status() { return name + " Life=" + health; }
}
public class L21Example02 {
    public static void main(String[] args) {
        Enemy e = new Enemy("Guard", 20);
        e.takeDamage(7);
        System.out.println(e.status());
    }
}
```

Output

```
Guard Life=13
```

Life 1 to 40, the initial life and damage of the subclass are all conventions of this lesson model, not Java restrictions, nor official game values. Allow health to drop to 0 after being injured.

04 / 15

extends to the family tree

extends Enemy means Skeleton is a type of Enemy. It can add its own fields and methods, and can also override the behavior of the parent class.

Subclasses automatically have inheritable members

Element	Explanation
extends	A statement is a relationship
super	Initialize parent class part
attack	Subclasses can override behavior

The subclass object contains the parent class part, but only one copy of the parent class code is maintained. After modifying takeDamage, all subclasses get the same fix.

L21Example03.java · Complete independent example

```
class Enemy {
    protected final String name;
    Enemy(String name) { this.name = name; }
    public String attack() { return name + " Normal attack"; }
}
class Skeleton extends Enemy {
    Skeleton(String name) { super(name); }
    @Override public String attack() { return name + " Archery"; }
}
public class L21Example03 {
    public static void main(String[] args) {
        Skeleton e = new Skeleton("Xiaobai");
        System.out.println(e.attack());
        System.out.println(e instanceof Enemy);
    }
}
```

Output

```
Xiaobai Archery
true
```

05 / 15

super completes parent class initialization

When creating a Skeleton, Java must first initialize its Enemy part. `super(...)` gives the name and life to the parent class constructor.

When a subclass is born, the parent class part is built first

Element	Explanation
Parent class constructor	Complete the common state first
Subclass constructor	Re-initialize exclusive fields
Matching parameters	Super() cannot be used by air

This chapter uses Java 21: explicit `super(...)` or `this(...)` is the first statement of the constructor. Here the parent class is initialized first, and then the subclass fields are initialized; the new rules of later versions of the language do not apply to this chapter.

L21Example04.java · Complete independent example

```
class Enemy {
    protected final String name;
    private int health;
    Enemy(String name, int health) {
        this.name = name == null || name.isBlank()
            ? "Unnamed" : name.strip();
        this.health = Math.max(1, Math.min(40, health));
    }
    public void takeDamage(int amount) {
        if (amount > 0) health = Math.max(0, health - amount);
    }
    public int health() { return health; }
    public String attack() { return name + " Normal attack 1"; }
    public String status() { return name + " Life=" + health; }
}
class Skeleton extends Enemy {
    private int arrows;
    Skeleton(String name) { this(name, 12); }
    Skeleton(String name, int arrows) {
        super(name, 20);
        this.arrows = Math.max(0, Math.min(64, arrows));
    }
    public int arrows() { return arrows; }
    @Override public String attack() { return name + " Archery 6"; }
}
public class L21Example04 {
    public static void main(String[] args) {
        Skeleton e = new Skeleton("Xiaobai", 99);
        System.out.println(e.status());
        System.out.println(e.arrows());
    }
}
```

Output

```
Xiaobai Life=20
64
```

Fault experiment: The parent class does not have a parameterless constructor; the implicit `super()` of the subclass cannot find a matching entry. Errors are intentionally left below, please anticipate them first and then try to fix them.

BadSuper.java · Intentional error example

```
class Enemy { Enemy(String name) { } }
class Skeleton extends Enemy { Skeleton() { } }
public class BadSuper { public static void main(String[] args) { } }
```

06 / 15

Override rewrite attack behavior

The parent class provides general attacks, and the subclasses can be rewritten according to their own fighting methods. When attack is called on a subclass object, the subclass version is executed.

The same method signature is replaced with a subclass version

Element	Explanation
Method signature	Name and parameters must match
@Override	Compiler check rewrite
Actual object	Determine instance method version

@Override is not decoration. When the method name or parameters are written incorrectly, it will cause the compiler to immediately alert you to prevent you from thinking that the rewrite was successful.

L21Example05.java · Complete independent example

```
class Enemy {
    protected final String name;
    Enemy(String name) { this.name = name; }
    public String attack() { return name + " Normal attack"; }
}
class Skeleton extends Enemy {
    Skeleton(String name) { super(name); }
    @Override public String attack() { return name + " Archery"; }
}
public class L21Example05 {
    public static void main(String[] args) {
        Enemy e = new Skeleton("Xiaobai");
        System.out.println(e.attack());
    }
}
```

Output

Xiaobai Archery

Fault experiment: attach is misspelled, @Override requires it to indeed override the parent class method. Errors are intentionally left below, please anticipate them first and then try to fix them.

BadOverride.java · Intentional error example

```
class Enemy { public String attack() { return "Normal attack"; } }  
class Skeleton extends Enemy {  
    @Override public String attack() { return "Archery"; }  
}  
public class BadOverride { public static void main(String[] args) { } }
```

07 / 15

Three kinds of enemies and three kinds of attacks

All three types of enemies inherit Enemy, but have different attack text and damage values. Differences remain in the subclass, and common life rules remain in the parent class.

Inheritance: The hollow triangle points to the parent class

Enemy

Subclass	Superclass	Relationship
Zombie	Enemy	Swing 4
Skeleton	Enemy	Archery 6
Creeper	Enemy	Explosion 9

Subclasses only override attack and do not copy takeDamage and status. The fewer copies, the easier the family tree is to maintain.

L21Example06.java · Paragraph 1/2, sequentially connected to the same file

```
class Enemy {
    protected final String name;
    private int health;
    Enemy(String name, int health) {
        this.name = name == null || name.isBlank()
            ? "Unnamed" : name.strip();
        this.health = Math.max(1, Math.min(40, health));
    }
    public void takeDamage(int amount) {
        if (amount > 0) health = Math.max(0, health - amount);
    }
    public int health() { return health; }
    public String attack() { return name + " Normal attack 1"; }
    public String status() { return name + " Life=" + health; }
}
class Zombie extends Enemy {
    Zombie(String name) { super(name, 22); }
    @Override public String attack() { return name + " Swing 4"; }
}
class Skeleton extends Enemy {
    private int arrows;
    Skeleton(String name) { this(name, 12); }
    Skeleton(String name, int arrows) {
        super(name, 20);
        this.arrows = Math.max(0, Math.min(64, arrows));
    }
    public int arrows() { return arrows; }
    @Override public String attack() { return name + " Archery 6"; }
```

L21Example06.java · Paragraph 2/2, sequentially connected to the same file

```
    }
    class Creeper extends Enemy {
        Creeper(String name) { super(name, 18); }
        @Override public String attack() { return name + " Explosion 9"; }
    }
    public class L21Example06 {
        public static void main(String[] args) {
            System.out.println(new Zombie("Carrion").attack());
            System.out.println(new Skeleton("Xiaobai").attack());
            System.out.println(new Creeper("Mung beans").attack());
        }
    }
}
```

Output

```
Carrion Swing 4
Xiaobai Archery 6
Mung beans Explosion 9
```

08 / 15

super method reuses parent class behavior

Sometimes the subclass does not completely replace the behavior of the parent class, but reuses it first and then adds its own effect.

First execute the parent class version and then add the subclass effect

Element	Explanation
super.attack()	Call parent class version
Supplementary effect	Add knockback after original result
attack()	Unconditional self-calling will be recursive

super.attack() calls the parent class version, and attack() may enter the current overridden method again. Wrong writing will cause infinite recursion.

L21Example07.java · Complete independent example

```
class Enemy {
    protected final String name;
    Enemy(String name) { this.name = name; }
    public String attack() { return name + " Normal attack"; }
}
class ArmoredZombie extends Enemy {
    ArmoredZombie(String name) { super(name); }
    @Override public String attack() {
        return super.attack() + " + repel";
    }
}
public class L21Example07 {
    public static void main(String[] args) {
        System.out.println(new ArmoredZombie("Iron armor").attack());
    }
}
```

Output

```
Iron armor Normal attack + repel
```

If super.attack() is replaced by attack(), the current method will call itself repeatedly until a StackOverflowError is thrown. Draw the calling route first, then type the code.

09 / 15

Access rights guard the boundaries

Subclasses are not the master key to parent classes. The private fields of the parent class are still not directly accessible, and the rules should be maintained through the methods provided by the parent class.

The subclass can be expanded but the parent class safe cannot be dismantled

Element	Explanation
private health	Independent subclasses cannot be accessed directly
public method	Provide controlled operation externally
protected name	Allow subclasses to read names

Use private first. Only expand permissions when subclasses really need to read directly; do not expose fields if you can use methods to solve them.

Fault experiment: Between these two independent classes, the subclass cannot directly use the private field of the parent class. Errors are intentionally left below, please anticipate them first and then try to fix them.

BadHealth.java · Intentional error example

```
class Enemy {
    private int health = 20;
    public int health() { return health; }
}
class Skeleton extends Enemy {
    void cheat() { health = 999; }
}
public class BadHealth { public static void main(String[] args) { } }
```

Change cheat to operate via a public controlled method. There are private access exceptions between Java's outermost class and its nested classes; this example intentionally uses a separate class to test the boundary.

Acceptance comparison table

Members	Suggested permissions	Reason
health	private	It is forbidden to bypass the rules of life
name	protected or private	Select according to expansion needs
takeDamage	In package or public	Provide controlled behavior
attack	In package or public	Allow overwriting and calling

10 / 15

How to choose inheritance and combination

Inherited expression type relationships, a combined expression object owns another object. Choosing the wrong relationship can make the family tree grow strange branches.

It is a kind of inheritance and there is a kind of combination

Element	Explanation
Skeleton is Enemy	Inherit connection type
Skeleton has Bow	Field expression possesses
Bow is not Enemy	Don't pick the wrong family tree

Read relationships into natural language. If you can only say "The skeleton has a bow," put Bow in the field and don't write extends Bow.

Acceptance comparison table

Problem	The answer is yes	Selection
Is Skeleton an Enemy?	Yes	Inheritance
Is Skeleton Bow?	No	Do not inherit
Does Skeleton have Bow?	Yes	Combination
Is the challenge Enemy?	Usually.	Inheritable

L21Example09.java · Complete independent example

```
class Enemy {
    protected final String name;
    Enemy(String name) { this.name = name; }
    public String attack() { return name + " Normal attack"; }
}
class Bow { String shoot() { return "Draw the bow"; } }
class Skeleton extends Enemy {
    private final Bow bow = new Bow();
    Skeleton(String name) { super(name); }
    @Override public String attack() { return name + " " + bow.shoot(); }
}
public class L21Example09 {
    public static void main(String[] args) {
        Skeleton e = new Skeleton("Xiaobai");
        System.out.println(e.attack());
    }
}
```

Output

```
Xiaobai Draw the bow
```

11 / 15

Three types of test family rules

Inheritance testing must not only check the differences between subclasses, but also confirm that the common rules of the parent class have not been violated.

Structure, attack, and injury all require checking boundaries.

Element	Explanation
Normal damage	Life is reduced on request
Negative damage	Keep the original value according to the contract
Extreme damage	The minimum life is 0

Each test first writes the initial state, then performs an action, and finally checks the output. Don't do five moves in a row and then guess which move was wrong.

Acceptance comparison table

Scene	Operation	Expected
Normal	Skeleton attack	Output Archery 6
Normal	Zombie injured 5	Life reduced by 5
Border	Suffering huge damage	Life stops at 0
Error	Negative damage	Life remains unchanged
Return	Three subclasses status	The format is consistent

L21Example10.java · Complete independent example

```
class Enemy {
    protected final String name;
    private int health;
    Enemy(String name, int health) {
        this.name = name == null || name.isBlank()
            ? "Unnamed" : name.strip();
        this.health = Math.max(1, Math.min(40, health));
    }
    public void takeDamage(int amount) {
        if (amount > 0) health = Math.max(0, health - amount);
    }
    public int health() { return health; }
    public String attack() { return name + " Normal attack 1"; }
    public String status() { return name + " Life=" + health; }
}
class Zombie extends Enemy {
    Zombie(String name) { super(name, 22); }
    @Override public String attack() { return name + " Swing 4"; }
}
public class L21Example10 {
    public static void main(String[] args) {
        Enemy e = new Zombie("Carrion");
        e.takeDamage(-5);
        System.out.println(e.health());
        e.takeDamage(Integer.MAX_VALUE);
        System.out.println(e.health());
    }
}
```

Output

```
22
0
```

12 / 15

Challenge Blueprint Enemy Family

The challenge program is called EnemyFamily. The parent class saves the name and life, and the three subclasses override attack. The complete program also demonstrates that the parent class injury rules are valid for all child classes.

Common status of parent class management and attack differences of sub-class management

Element	Explanation
Parent class	Save common state
Three subcategories	Implement attacks separately
Call program	Testing Differences and Common Rules

First write Enemy and test status, then add a subclass. After the first subclass is cleared, copy the minimum skeleton to create the other two.

Acceptance sequence

- Create zombies, skeletons and creepers.
- Three different attacks are output in sequence.
- Let the skeleton be injured by 7 points and output the status.
- Inflict negative damage to the creeper, and its status must remain unchanged.
- Confirm that all three subclasses are Enemy.

13 / 15

Challenge Realizing the Enemy Family

Create new EnemyFamily.java. This program only uses standard Java 21 and can be compiled and run directly.

Complete program verification super rewriting and parent class rules

Element	Explanation
EnemyFamily.java	The auxiliary class and the main class are the same file
One action at a time	Check Injury and Status
Inheritance evidence	instanceof is true

Explain the role of super in the three constructors; deliberately misspell attack and retain @Override; confirm that the compiler finds the problem before running.

EnemyFamily.java · Section 1/2, sequentially connected to the same file

```
class Enemy {
    protected final String name;
    private int health;
    Enemy(String name, int health) {
        this.name = name == null || name.isBlank()
            ? "Unnamed" : name.strip();
        this.health = Math.max(1, Math.min(40, health));
    }
    public void takeDamage(int amount) {
        if (amount > 0) health = Math.max(0, health - amount);
    }
    public int health() { return health; }
    public String attack() { return name + " Normal attack 1"; }
    public String status() { return name + " Life=" + health; }
}
class Zombie extends Enemy {
    Zombie(String name) { super(name, 22); }
    @Override public String attack() { return name + " Swing 4"; }
}
class Skeleton extends Enemy {
    private int arrows;
    Skeleton(String name) { this(name, 12); }
    Skeleton(String name, int arrows) {
        super(name, 20);
        this.arrows = Math.max(0, Math.min(64, arrows));
    }
    public int arrows() { return arrows; }
    @Override public String attack() { return name + " Archery 6"; }
```

EnemyFamily.java · Section 2/2, sequentially connected to the same file

```
}  
class Creeper extends Enemy {  
    Creeper(String name) { super(name, 18); }  
    @Override public String attack() { return name + " Explosion 9"; }  
}  
public class EnemyFamily {  
    public static void main(String[] args) {  
        Zombie zombie = new Zombie("Captain Carrion");  
        Skeleton skeleton = new Skeleton("Xiaobai");  
        Creeper creeper = new Creeper("Mung beans");  
        System.out.println(zombie.attack());  
        System.out.println(skeleton.attack());  
        System.out.println(creeper.attack());  
        skeleton.takeDamage(7);  
        creeper.takeDamage(-3);  
        System.out.println(skeleton.status());  
        System.out.println(creeper.status());  
        System.out.println(skeleton instanceof Enemy);  
    }  
}
```

Output

```
Captain Carrion Swing 4  
Xiaobai Archery 6  
Mung beans Explosion 9  
Xiaobai Life=13  
Mung beans Life=18  
true
```

14 / 15

Expansion tasks and clearance review

The assignment must draw a class diagram arrow and write a sentence to prove that each inheritance relationship is one.

The third-level task continues to expand the family tree

Element	Explanation
Class diagram	The arrow points from the subclass to the parent class
Test table	Normal, Boundary, Error
Bug records	Write down the reasons and minimal fixes

Lesson 22 lets Enemy reference unified commands for different subcategories. The same attack command automatically selects the version of the real object.

Third gear operation

- Wooden pickaxe: Create two subclasses, Zombie and Skeleton, and rewrite attack respectively.
- Iron pickaxe: Add move to the parent class, and the subclass only rewrites the truly different movement methods.
- Diamond Pickaxe: Design a two-stage challenge. In the second stage, super.attack is reused and special effects are added.

Acceptance comparison table

Self-evaluation	Can	Still need to practice
I can determine the inheritance relationship	☐	☐
I would use super	☐	☐
I will rewrite the method correctly	☐	☐

15 / 15

Own acceptance record

Beautiful output is the starting point, and being able to explain boundaries is considered mastery. Select one case each from normal, borderline and error, write down your expectations before running, and leave evidence after running.

Leave a basis for judgment

Category	The operation you selected	Check key points
Normal	A common input or a successful action	Is the result consistent line by line?
Border	Range endpoints and adjacent values	Is there one more step or one less thing?
Error	Illegal quantity, missing object or bad data	Whether it is explicitly rejected and whether the status is maintained

- Bug records: triggering conditions, actual phenomena, causes, and minimum repairs.

- Illustrated record: describe the starting point, end point and meaning of each arrow.
- Adaptation record: only change one rule, and modify expectations and tests at the same time.

Technical check: Java SE 21 language specification and standard library. See the language specification for class, interface, type and expression rules; see the corresponding API for the behavior of collections, Random, Files and Stream.

<https://docs.oracle.com/javase/specs/jls/se21/html/index.html>

<https://docs.oracle.com/en/java/javase/21/docs/api/index.html>