

JAVA / LESSON 18 OF 40

An Army of Objects

Reference object interaction and null

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

Create player and pet interactive team

Will understand object variables as references; can explain the effect of two variables pointing to the same object; can handle null before using the object; can allow players to interact safely with pets through parameters.

Prerequisites: declares classes, creates objects, accesses fields and calls instance methods. Today we need to see clearly where the "coordinate paper" of the object variable points.

Before you start: It is recommended to complete Lesson 17 first

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=18&lang=en>

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

Two tags may find the same wolf

Element	Explanation
#1 · Pet object	name = shadow health = 17

Variable	Refers to
shadow	#1
alias	#1

In this lesson

01. Before you begin: Put the examples in the right place
02. Reference image object coordinates
03. Two variables share an object
04. null means there is no object
05. Pass the object to the method
06. Objects interact through methods
07. Changing the reference is different from changing the object
08. Reference equality and content equality
09. Object Legion Test Bench
10. null Fault Clinic
11. Challenge Blueprint Players and Pets
12. Challenge Realization Player Pet Team
13. Expansion tasks and clearance review
14. Own acceptance record

01 / 14

Before you begin: Put the examples in the right place

This chapter uses JDK 21 and does not require the preview function. A complete program is given for each common example, saved with annotated file names. Multiple examples are placed in their own folders to avoid auxiliary classes with the same name.

Terminal command: Replace the Chinese placeholder with the actual name

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

Class names are case-sensitive. If the complete challenge code is divided into multiple sections, please connect them in the same file in order. Do not create separate classes for the continued sections. Run verification only lists the program output; additional input sequences are provided where input is required.

Learning action: first write down your prediction, compare it after running, and then change only one condition. Experiments marked "Intentional Bug" are used to identify bugs; they are clearly distinguished from ready-to-run examples.

02 / 14

Reference image object coordinates

Object variables do not directly hold all data like int. It holds a reference that allows programs to find the object and access fields or methods.

The object variable stores a reference

Element	Explanation
#1 · Pet object	name = shadow health = 20

Variable	Refers to
wolf	#1

L18Example01.java · Complete independent example

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
}
public class L18Example01 {
    public static void main(String[] args) {
        PetWolf wolf = new PetWolf();
        wolf.name = "Shadow";
        System.out.println(wolf.name);
    }
}
```

Output

Shadow

Reading route

- new PetWolf() creates the object.
- wolf holds a reference to this object.
- wolf.name follows the reference to find the field.
- wolf.heal(3) calls the method along the reference.

It is not a copy of the box: when assigning wolf to another object variable, a new wolf will not be copied by default, but only a note pointing to the original object.

03 / 14

Two variables share an object

Reference assignment copies the reference. first and second point to the same object, so after modifying health through second, first reads the new value.

Two references can point to the same pet

Element	Explanation
#1 · Pet object	name = shadow health = 1

Variable	Refers to
first	#1
second	#1

Arrows indicate reference relationships; objects are not automatically copied.

L18Example02.java · Complete independent example

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
}
public class L18Example02 {
    public static void main(String[] args) {
        PetWolf first = new PetWolf();
        PetWolf second = first;
        second.health = 1;
        System.out.println(first.health);
        System.out.println(first == second);
    }
}
```

Output

```
1
true
```

Detective problem: If you originally wanted to copy the pet, reference assignment did not do it. New objects can be created explicitly later using a constructor or copy method.

04 / 14

null means there is no object

The object variable may not point to any object temporarily, and the value is null at this time. If the left side of the dot is null, the program cannot find the field or method.

Null reference has no coordinates to go to

Element	Explanation
null	There is currently no object
Check	First determine whether it is non-null
Exist	Call object method
missing	Prompt or return failure

L18Example03.java · Complete independent example

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
}

public class L18Example03 {
    public static void main(String[] args) {
        PetWolf selected = null;
        if (selected != null) System.out.println(selected.name);
        else System.out.println("No pet selected yet");
    }
}
```

Output

No pet selected yet

Empty box prompt: null is not an empty string, nor is it an object with a life of 0. It means that no objects are currently available.

05 / 14

Pass the object to the method

Objects can be passed into methods as parameters. What is passed in is still a reference, so the method modifies the object state, and the caller can see the change.

Method parameters can also receive object references

Element	Explanation
Actual parameters	Pass in a reference value
Formal parameters	Receive a copy of the reference value
Change object	The caller sees the status change
Change parameters	Do not change the caller's reference

Java is always pass by value. The object parameter gets a copy of the reference value: modifications to the same object along the copy are visible to the caller; changing the parameter to point to a new object does not change the caller's wolf variable.

Parameter check: When the public helper method receives an object, it first decides whether null is allowed. If it is allowed, deal with it clearly, and if it is not allowed, reject it as soon as possible.

L18Example04.java · Complete independent example

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
}
public class L18Example04 {
    static void healPet(PetWolf pet, int n) {
        if (pet != null) pet.heal(n);
    }
    static void replacePet(PetWolf pet) { pet = new PetWolf(); }
    public static void main(String[] args) {
        PetWolf wolf = new PetWolf(); wolf.health = 12;
        healPet(wolf, 5);
        replacePet(wolf);
        healPet(null, 5);
        System.out.println(wolf.health);
    }
}
```

Output

17

06 / 14

Objects interact through methods

Object interaction is not about letting one class change the fields of another class casually. A clearer way is to call the other party's method.

The player takes action and the pet maintains its own status

Element	Explanation
Player	Responsible for selecting interactive objects
feed	Call pet eat
PetWolf	Maintain trust scope
missing	Handle null explicitly

L18Example05.java · Complete independent example

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
}
class Player {
    void feed(PetWolf pet) { if (pet != null) pet.eat(); }
}
public class L18Example05 {
    public static void main(String[] args) {
        Player alex = new Player();
        PetWolf pet = new PetWolf();
        alex.feed(pet);
        alex.feed(null);
        System.out.println(pet.trust);
    }
}
```

Output

10

Knock access: Player does not write `pet.trust += 10` directly, but calls `pet.eat()`. The pet itself maintains the upper limit of trust value.

07 / 14

Changing the reference is different from changing the object

`second.health = 5` modifies the object; `second = new PetWolf()` changes the reference saved by `second`. `first` still points to the old object.

Reassignment only changes that reference

Element	Explanation
#1 · Object A	health = 20
#2 · Object B	health = 5

Variable	Refers to
first	#1
second	#2

L18Example06.java · Complete independent example

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
}
public class L18Example06 {
    public static void main(String[] args) {
        PetWolf first = new PetWolf();
        PetWolf second = first;
        second = new PetWolf();
        second.health = 5;
        System.out.println(first.health + "," + second.health);
        System.out.println(first == second);
    }
}
```

Output

```
20,5
false
```

Draw arrows: Draw object squares and variable arrows when encountering reference questions. Once the arrow is drawn, it is difficult for the Enderman to teleport away logically.

08 / 14

Reference equality and content equality

Using `==` on an object compares whether two references point to the same object. Two pets with the same name may still be different people.

Whether double equal sign comparison points to the same object

Element	Explanation
==	Compare citation identities
sameName	This example compares name content
Same name	Does not represent the same object
equals	Whether rewriting will affect semantics

L18Example07.java · Complete independent example

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
    boolean sameName(PetWolf other) {
        return other != null && java.util.Objects.equals(name, other.name);
    }
}

public class L18Example07 {
    public static void main(String[] args) {
        PetWolf a = new PetWolf(); a.name = "Shadow";
        PetWolf b = a;
        PetWolf c = new PetWolf(); c.name = "Shadow";
        System.out.println(a == b);
        System.out.println(a == c);
        System.out.println(a.sameName(c));
    }
}
```

Output

```
true
false
true
```

Don't be confused: String's equals compares text content; if ordinary custom objects do not override equals, they still use reference identity rules.

09 / 14

Object Legion Test Bench

When testing the object program, record which object the variable points to and the current field value of each object.

Testing reference graphs is more reliable than just looking at the output

Element	Explanation
new twice	Verify independent status
Shared assignment	Verify that modifications are visible
Redirect	Verify that old references have not changed
null	Verification Security Rejection

Acceptance and inspection forms

Scene	Operation	Expected
Independent	new twice	The two objects do not affect each other
Share	b = a	After b is modified, a can also see
Change quote	b = new PetWolf()	a still points to the old object
Null reference	b = null	Do not call method after checking

Evidence first: don't just say "should be the same". Write the result of `a==b` and print the health values of the two objects.

10 / 14

null Fault Clinic

NullPointerException usually gives a specific line number. Find all dots in this line and check the source of each object variable.

Find the first dangerous point number along the exception stack

Element	Explanation
Look at the line number	Locate your own code
Find references	Left side of checkpoint number
Check the source	Track assignment and return
Repair	Arrange branches for missing objects

L18Example09.java · Complete independent example

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
}
public class L18Example09 {
    static PetWolf findPet(String name) { return null; }
    public static void main(String[] args) {
        PetWolf found = findPet("Shadow");
        if (found != null) found.heal(5);
        else System.out.println("There is no such pet");
    }
}
```

Output

```
There is no such pet
```

This example uses the minimum search function of "always not found" to test the missing branch. The real selection system has to search in the pet list; if it cannot be found, it returns null and the caller handles it explicitly.

Emergency sequence

- Read the exception type and the first line of your own code.

- Circle the object variable to the left of the dot.
- Track where it is assigned or returned from.
- Choose to check for null, provide a default object, or reject input.

Don't swallow exceptions: Don't pretend to fix them with empty catches. The real fix is to make object missing an explicit branch.

11 / 14

Challenge Blueprint Players and Pets

The challenge program is called PlayerPetTeam. The player holds the current pet reference; it cannot be fed or treated until the selection is successful.

Three object methods to complete feeding, treatment and selection

Element	Explanation
Player.selected	Save current selection
PetWolf	Maintain life and trust
Interaction	Call pet method
Cancel	null clear selection

Acceptance conditions

- The two pets are independent objects.
- After the player selects the first pet and feeds it, the trust value increases.
- Point another variable to the same pet and treat it, and the original reference sees new life.
- After deselecting is set to null, the interaction returns false instead of crashing.

Acceptance and inspection forms

Method	Enter	Output or effect
select	PetWolf	Save current reference
feedSelected	None	Trust increased or false
healSelected	Quantity	Life increased or false
status	None	Print pet status

Interface division of labor: Player decides who to choose; PetWolf decides whether the status after eating and treatment is legal.

12 / 14

Challenge Realization Player Pet Team

Create new PlayerPetTeam.java. Predict each line of output before running, and then draw the objects pointed to by alex.selected, alias, shadow, and snow.

Reference relationships become verifiable behavior in the complete program

Element	Explanation
Select shadow	selected points to shadow
Feeding	Trust increased by 10
alias treatment	The original object's life changes to 17
Cancel	Interaction returns false

Acceptance: Explain why alias can change shadow; confirm that snow is not affected; the program still ends normally after deselecting.

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

```
public class PlayerPetTeam {
    static class PetWolf {
        String name;
        int health = 20;
        int trust = 0;
        void eat() { trust += Math.min(10, 100 - trust); }
        void heal(int n) {
            if (n > 0) health += Math.min(n, 20 - health);
        }
        void status() {
            System.out.println(name + " Life=" + health + " Trust=" + trust);
        }
    }
    static class Player {
        String name;
        PetWolf selected;
        void select(PetWolf pet) { selected = pet; }
        boolean feedSelected() {
            if (selected == null) return false;
            selected.eat(); return true;
        }
        boolean healSelected(int n) {
            if (selected == null || n <= 0) return false;
            selected.heal(n); return true;
        }
    }
    public static void main(String[] args) {
        PetWolf shadow = new PetWolf(); shadow.name = "Shadow";
```

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

```
        PetWolf snow = new PetWolf(); snow.name = "Snowball";
        shadow.health = 12;
        Player alex = new Player(); alex.name = "Alex";
        alex.select(shadow);
        System.out.println(alex.feedSelected());
        PetWolf alias = shadow;
        alias.heal(5);
        shadow.status();
        snow.status();
        alex.select(null);
        System.out.println(alex.healSelected(3));
    }
}
```

Output

```
true
Shadow Life=17 Trust=10
Snowball Life=20 Trust=0
false
```

13 / 14

Expansion tasks and clearance review

In addition to the code, the homework also requires a reference diagram: variables are drawn as labels, objects are drawn as squares, and arrows are used to connect them.

Three-speed task training object arrow

Element	Explanation
Wooden Pickaxe	Prove that shared modifications are visible
Iron Pickaxe	Prove that parameters are passed by value
Diamond Pickaxe	Three pet selection system
Review	Draw variable and object arrows

Third gear operation

- Wooden Pickaxe: Create a Mob, let two variables share it and prove that modifications are visible.
- Iron Pick: Write the method `heal(Mob target, int amount)`, safely handle null, and prove that reassigning parameters does not change the caller's reference.
- Diamond Pickaxe: Make a selection system for three pets, supporting selection, cancellation, feeding and status query.

Acceptance and inspection forms

Self-evaluation	Can	Still need to practice
I can draw reference arrows	☐	☐
I can explain shared objects	☐	☐
I will handle null safely	☐	☐

Next lesson: Lesson 19: Study the generation point of the object. The constructor will complete initialization and the first round of data check when the object is born.

14 / 14

Own acceptance record

Write expectations before running, and record actual results after running. Normal cases show that the program does things; edge cases show that it knows when to stop; error cases show that it rejects unreasonable requests.

Write the evidence into the record

Category	Input of your choice	Expectation and actual
Normal	A common value or a successful operation	Is it consistent line by line?
Border	Minimum value, maximum value and adjacent value	Is there one less square and one more step?
Error	Illegal quantity, missing object or incorrect syntax	Explicitly reject? Is the status maintained?

- Write a bug as: triggering conditions, phenomena, causes, and minimal repairs.
- Organize a rule worth remembering and explain it in your own words. Don't just copy the definition.
- Change an example, keep a record of the test, and let the results prove your understanding.

Technical verification basis: Java SE 21 language specification, scope and access control (Chapter 6), classes and constructors (Chapter 8), method invocation (Chapter 15).

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