

JAVA / LESSON 19 OF 40

Spawn Points and Constructors

Constructor this and initialization

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

Make three kinds of entrance player generators

Can write a constructor with the same name as the class; can use this to distinguish fields and parameters; can complete verification when the object is born; can use constructor overloading to provide three legal birth methods.

Prerequisites: classes, fields, instance methods, parameters, overloads and object references. Today, every object must be in a legal state from the first second of birth.

Before you start: It is recommended to complete Lessons 17-18 first

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=19&lang=en>

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

Three entrances, one initialization route

Step	Explanation
1. Default born	this("Steve")
2. Specify name	this(name, 20)
3. Complete constructor	Verify and write fields

In this lesson

01. Before you begin: Put the examples in the right place
02. constructor is responsible for birth physical examination
03. Constructor syntax map
04. this points to the current object
05. Complete verification at birth
06. Constructor overloading
07. this constructor chain
08. Legal status and testing
09. Constructor Failure Clinic
10. Challenge Realization Three Legal Births
11. Expansion tasks and clearance review
12. Own acceptance record

01 / 12

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 / 12

constructor is responsible for birth physical examination

If you write new first and then write fields one by one, it is easy to miss names or leave negative lives. The constructor puts the necessary data into the creation process.

new calls the constructor to create and initialize the object

Element	Explanation
new	Select matching constructor
Constructor	Initialize new object
Field	Save birth status
Authentication	Ensure legality when creation is completed

L19Example01.java · Complete independent example

```
class Player {
    String name;
    int health;
    Player(String name, int health) {
        this.name = name == null || name.isBlank() ? "Steve" : name.trim();
        this.health = Math.max(1, Math.min(20, health));
    }
    Player(String name) { this(name, 20); }
    Player() { this("Steve"); }
}

public class L19Example01 {
    public static void main(String[] args) {
        Player alex = new Player("Alex", 20);
        System.out.println(alex.name + ", " + alex.health);
    }
}
```

Output

```
Alex, 20
```

Identification method: The constructor has the same name as the class and has no return type. If written as `void Player`, it will become a normal method and will not be automatically called when `new`.

03 / 12

Constructor syntax map

The constructor is written in the class body. The actual parameters after `new` must match a constructor parameter list.

Class name, parameters and constructor body are indispensable.

Element	Explanation
Class name	The constructor must have the same name
Parameter list	Must match actual parameters
Return type	Constructor has no return type
No parameters	After explicitly declaring the constructor, no parameters will be automatically filled.

L19Example02.java · Complete independent example

```
class Mob {
    String name; int health;
    Mob(String name, int health) {
        this.name = name; this.health = health;
    }
}
public class L19Example02 {
    public static void main(String[] args) {
        Mob zombie = new Mob("Small stiffness", 20);
        System.out.println(zombie.name);
    }
}
```

Output

```
Small stiffness
```

Automatic changes: Once a parameterized constructor is declared, Java no longer automatically provides a parameterless constructor. `new Mob()` will not find the entry.

04 / 12

this points to the current object

Constructors often make parameter names consistent with field names. At this time, the short name represents the parameter, and the `this.` field represents the object field being created.

When the parameter name covers the field name, this opens the object box

Element	Explanation
<code>this.name</code>	Current object field
<code>name</code>	The parameters passed in this time
Correct assignment	Field receives parameter value
<code>name=name</code>	The parameters are assigned to itself, and the fields remain unchanged.

L19Example03.java · Complete independent example

```
class Player {
    String name;
    Player(String name) { name = name; }
}
public class L19Example03 {
    public static void main(String[] args) {
        Player p = new Player("Alex");
        System.out.println(p.name);
    }
}
```

Output

```
null
```

Here `name = name` refers to parameters on both sides, and the field is still null. After changing to `this.name = name`, the output should be Alex; successful compilation may also hide logic errors.

Read the equal sign: `name` on the right side of the equal sign is the material sent; `this.name` on the left side is the grid in the object.

05 / 12

Complete verification at birth

The constructor is the first door to keep the legal state of the object. Define the rules first, then choose to modify values or throw exceptions. Safe defaults can be used during the initial learning phase.

Illegal material is corrected or rejected at constructor entry

Element	Explanation
Bad name	Replaced by Steve
Valid name	Remove blanks at both ends
Life	Corrected to 1 to 20
Strategy	All entrances remain consistent

L19Example04.java · Complete independent example

```
class Player {
    String name;
    int health;
    Player(String name, int health) {
        this.name = name == null || name.isBlank() ? "Steve" : name.trim();
        this.health = Math.max(1, Math.min(20, health));
    }
    Player(String name) { this(name, 20); }
    Player() { this("Steve"); }
}
public class L19Example04 {
    public static void main(String[] args) {
        Player p = new Player("", -8);
        System.out.println(p.name + "," + p.health);
    }
}
```

Output

```
Steve,1
```

Determine the strategy in advance: verification failure can be corrected, rejected or reported. A class should use a consistent strategy and not change secretly sometimes or suddenly crash sometimes.

06 / 12

Constructor overloading

The constructor can be overloaded to provide a concise entry for common creation methods. Parameter lists must be different.

Constructor delegation: only one copy of the verification is retained in the end

Step	Explanation
1. Player()	Provide default name
2. Player(name)	Provide default life
3. Complete constructor	Write field after verification

L19Example05.java · Complete independent example

```
class Player {
    String name;
    int health;
    Player(String name, int health) {
        this.name = name == null || name.isBlank() ? "Steve" : name.trim();
        this.health = Math.max(1, Math.min(20, health));
    }
    Player(String name) { this(name, 20); }
    Player() { this("Steve"); }
}

public class L19Example05 {
    public static void main(String[] args) {
        Player a = new Player("Alex", 12);
        Player b = new Player("Builder");
        Player c = new Player();
        System.out.println(a.name + "," + a.health);
        System.out.println(b.name + "," + b.health);
        System.out.println(c.name + "," + c.health);
    }
}
```

Output

```
Alex,12
Builder,20
Steve,20
```

Not three classes: all three constructors create Player, just the materials provided at birth are different.

07 / 12

this constructor chain

this(...) calls another constructor of this class, which must be the first statement of the constructor. In this way, only one copy of the default value and verification is maintained.

Constructor delegation: only one copy of the verification is retained in the end

Step	Explanation
1. Player()	Provide default name
2. Player(name)	Provide default life
3. Complete constructor	Write field after verification

Failure experiment: In Java 21 used in this course, this(...) must be the first statement of the constructor. Errors are intentionally left below, please anticipate them first and then try to fix them.

BadConstructorOrder.java · Intentional error example

```
class Player {  
    Player(String name) { }  
    Player() {  
        System.out.println("Birth");  
        this("Steve");  
    }  
}  
public class BadConstructorOrder {  
    public static void main(String[] args) { }}
```

Single route: Let all overloads eventually go to the most complete constructor. Once the verification is corrected, the three spawn points will become safe together.

08 / 12

Legal status and testing

The rules that are expected to be maintained in the design are called invariant conditions. This lesson first ensures that when the Player is first created, its name cannot be empty, and its life must be between 1 and 20 when it is born.

Objects should satisfy indestructible rules as soon as they are born

Element	Explanation
Name	Not empty after creation
Life	1 to 20 at birth
coordinates	Within project -100 to 100
Long-term guarantee	It is also necessary to encapsulate to prevent arbitrary modifications

Acceptance and inspection forms

Test	Creation method	Expected
Complete	<code>new Player("Alex", 12)</code>	Alex 12
Default life	<code>new Player("Alex")</code>	Alex 20
Full default	<code>new Player()</code>	Steve 20
Bad name	<code>new Player("", 8)</code>	Steve 8
Bad life	<code>new Player("Alex", 99)</code>	Alex 20

Boundary values: Life test contains at least 0, 1, 20, 21. It's easier to catch errors on both sides of the boundary than just measuring 10.

09 / 12

Constructor Failure Clinic

When reading "constructor cannot be applied", compare the actual parameter quantity and type of new, and then check the constructor declaration.

Compilation errors usually come from name parameters or this order

Element	Explanation
void class name	Ordinary method, not constructor
Multiple actual parameters	No matching entry
this order	The first sentence is required in Java21
Missing field	May leave default value

Fault experiment: There is no constructor matching the three actual parameters. Errors are intentionally left below, please anticipate them first and then try to fix them.

`void Player` is a normal method. There is still an automatically provided no-argument constructor, so `new Player()` can be executed, but this ordinary method will not be automatically called.

Emergency sequence: check the class name; confirm that there is no return type; match parameters; check whether this(...) is the first sentence; finally verify that each field has been initialized.

BadConstructorCall.java · Intentional error example

```
class Player {
    Player(String name) { }
}
public class BadConstructorCall {
    public static void main(String[] args) {
        Player p = new Player("Alex", 20, 3);
    }
}
```

L19Example08B.java · Complete independent example

```
class Player {
    String name;
    void Player(String name) { this.name = name; }
}
public class L19Example08B {
    public static void main(String[] args) {
        Player p = new Player();
        System.out.println(p.name);
    }
}
```

Output

```
null
```

10 / 12

Challenge Realization Three Legal Births

The challenge program is called PlayerSpawn. The three constructors support complete birth, specified name and complete default respectively.

The constructor chain sends all birth modes into the same set of checks

Element	Explanation
Complete player	Name, life and coordinates
Specify name	The rest use default values
Default player	Steve and the default state
Bad material	Modify according to unified strategy

Acceptance: Explain which constructor is called by new four times respectively; confirm that the bad material is corrected; and then add boundary coordinates of -100 and 100.

PlayerSpawn.java · Complete standalone example

```
public class PlayerSpawn {
    static class Player {
        String name;
        int health;
        int x;
        int z;

        Player(String name, int health, int x, int z) {
            this.name = name == null || name.isBlank() ? "Steve" : name.trim();
            this.health = Math.max(1, Math.min(20, health));
            this.x = Math.max(-100, Math.min(100, x));
            this.z = Math.max(-100, Math.min(100, z));
        }
        Player(String name) {
            this(name, 20, 0, 0);
        }
        Player() {
            this("Steve");
        }
        void describe() {
            System.out.println(name + " Life=" + health
                               + " Birth point=(" + x + "," + z + ")");
        }
    }

    public static void main(String[] args) {
        Player a = new Player("Alex", 18, 12, -7);
        Player b = new Player("Builder");
        Player c = new Player();
        Player d = new Player("", -5, 999, -999);
        a.describe(); b.describe(); c.describe(); d.describe();
    }
}
```

Output

```
Alex Life=18 Birth point=(12,-7)
Builder Life=20 Birth point=(0,0)
Steve Life=20 Birth point=(0,0)
Steve Life=1 Birth point=(100,-100)
```

The rules only guarantee the birth state when construction is complete. At present, fields can still be modified by external code; in order to keep the rules throughout the entire use of the object, encapsulation in the next lesson is required. The life span of 1 to 20 at birth is the convention of this project; the injured object can be set to 0 to indicate falling down.

11 / 12

Expansion tasks and clearance review

The assignment requires writing at least two tests for each constructor and stating the invariant conditions of the object.

Third gear task expansion generation point

Element	Explanation
Wooden Pickaxe	Mob adds constructor
Iron Pickaxe	Overloading and Life Boundary
Diamond Pickaxe	Limit random birth area
Review	Birth rules and this

Third gear operation

- Wooden Pickaxe: Add a constructor to Lesson 17 Mob, and a name and life must be provided when creating it.
- Iron Pickaxe: Add overloaded constructor and parameter verification, keep life from 0 to 20.
- Diamond Pickaxe: Designed with random birth coordinates, but guaranteed not to fall within the designated danger range.

Acceptance and inspection forms

Self-evaluation	Can	Still need to practice
I can identify constructors	☐	☐
I would use this	☐	☐
I will test the birth rules	☐	☐

Next lesson: Lesson 20 Locking objects. Private protects fields, while public methods provide controlled operations like box buttons.

12 / 12

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>