Self-Learning Material #3

Loops

Overview

- Basic of Looping
- `while` loop
- `do-while` loop
- `for` loop
- Enhanced `for` loop
- `break` & `continue`
- Nested Loop

Basics of Looping

- To execute code *repeatedly*.
- Depends on the value of a boolean **condition**:
    - condition is `true`, loop continues
    - condition is `false`, loop ends
- Similar to `if` code can be a single line or many lines enclosed in curly braces `{}`
- Section of code executed is typically called loop body

Basic of Looping

- Four types of loop in Java
  - `while` loop
  - `do-while` loop
  - `for` loop
  - Enhanced `for` loop
- Some loops execute the body first and test the condition
- Some loops test the condition first and run the loop
- Some loops run for a predetermined number of iterations
- Some loops run for a unknown number of iterations
- Some never ends!

while loop

- `while` loop and `do-while` loop can execute body for undetermined number of times - based on a `boolean` loop condition

while loop

```
while (<loop condition>) {
    <loop body>
}
```

do-while loop

```
do {
    <loop body>
} while (<loop condition>);
```

while-loop

```
while (<loop condition>) {
    <loop body>
}
```

- Execute body while condition is **true**
- Loop condition tested before executing body
- If loop condition is false first time through, the body is not executed at all.

```
int x = 99; //or any random number
boolean loveme = true;
while (x > 1) {
    x = x / 2;
    System.out.println("She " + (loveme ? "loves me" : "does not love me"));
    loveme = !loveme;
}
```

do-while loop

```
do {
    <loop body>
} while (<loop condition>);
```

- Always executes loop body at least once
- Switches the order of test and loop body
- Loops back if the condition is true

```
char grade;
do {
    System.out.println("Take COMP2026");
    grade = result();
} while (grade == 'F');
```

🧑‍🏫 💬 Don't forget the `;` after the while!

while and do-while

- When evaluating the loop condition, follow the same rules as the conditions for `if-else` statements
- Multiple conditions can be combined using logical operators: `&&`, `||`, `!`, e.g.
  - `(numClasses >=3 ) && (numClasses <= 5)`
  - `(peopleStanding <= maxPeople) || (maxPeople < 50)`

while

- May not execute at all
- loop condition tested before loop body
- loop condition variables must be set before loop entry

do-while

- body always executed at least once
- loop condition tested at the bottom of loop
- the keyword `while` always has `;`

while loop and do-while loop

Both loops...

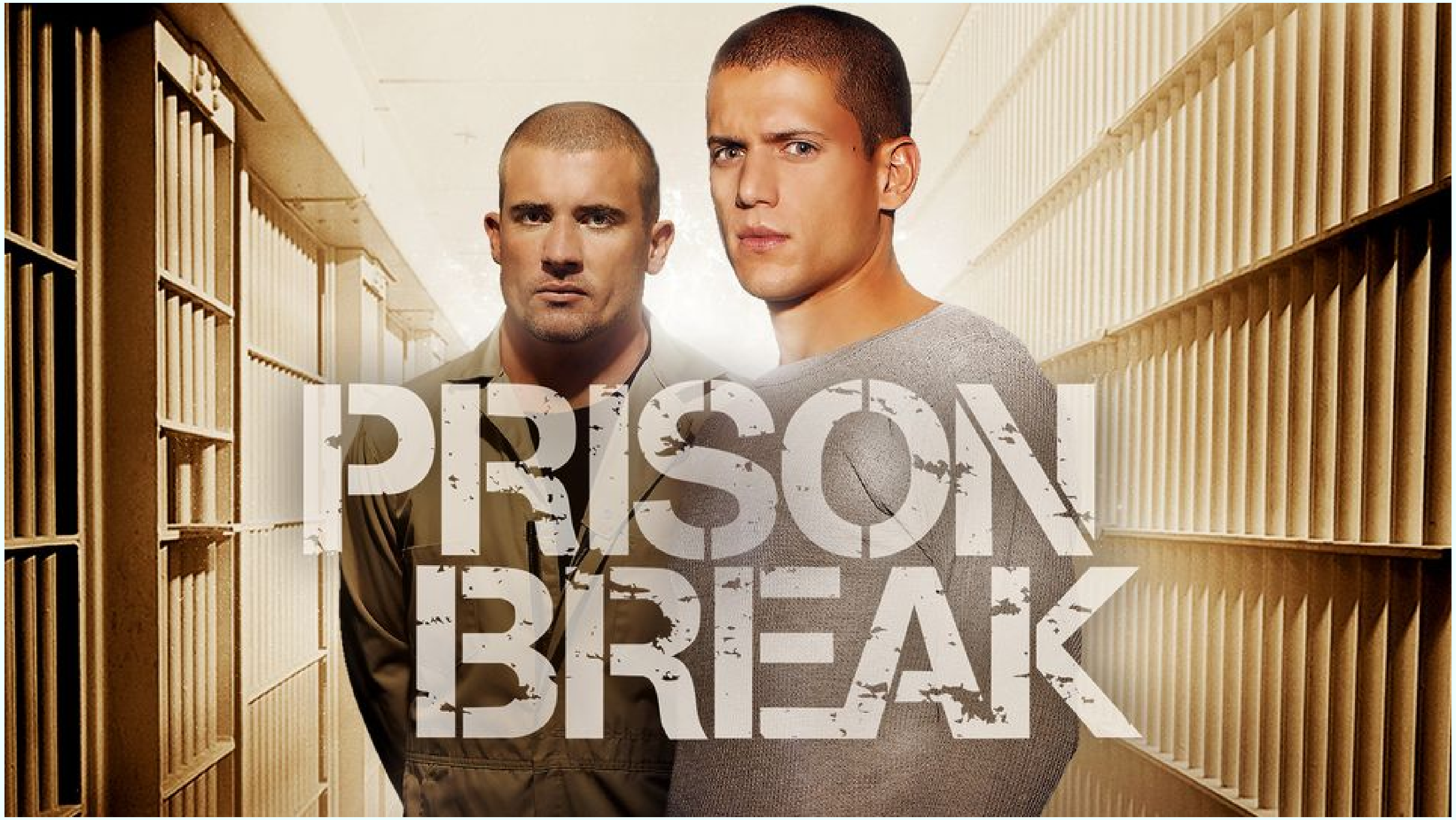
- stops executing body if loop condition is false
- must make sure loop condition becomes false by some computations to avoid an **infinite loop**.
- An infinite loop will never turn the condition to false - i.e., exit condition never occurs (and your program may "freeze up"!)

Infinite Loop

```
int days = 10;
while (days > 0) {
    System.out.printf("I need to stay here for another %d days\n", days);
    days--;
    if (days == 1) {
        System.out.println("Have a farewell party with cellmates");
        SYstem.out.println("Get caught and extend the penalty");
        days += 10;
    }
}
```

- That makes you stay in a prison forever.

img credit: https://www.tripadvisor.com/LocationPhotoDirectLink-g295370-d7815345-i145922401-RoomEscape_Split_by_Fox_in_a_box-Split_Split_Dalmatia_County_Dalmatia.html



img credit: Prison Break Facebook page

break Statement

- Recall `break` statement appears in a `switch` or a **loop**
- When used inside a `switch`, a `break` will jump to the end of the switch and execute the rest of the method
- When used inside a **loop**, a `break` will **exit the loop immediately** and execute the rest of the method.
- All four loops (while/do-while/for/enhanced for) support `break`.

break Statement

```java
int days = 10;
while (days > 0) {
    System.out.printf("I have been here for %d days\n", days);
    days--;
    if (days == 1) {
        System.out.println("Have a farewell party with cellmates");
        SYstem.out.println("Get caught and extend the penalty");
        days += 10;
    }
    if (days == 8)
        break;
}
System.out.println("I am out of here!");
```

```
I have been here for 10 days.
I have been here for 9 days.
I have been here for 8 days.
I am out of here!
```

continue Statement

- Used in a loop that skips the remaining statements in the body of the loop and proceeds with the next iteration.
- `continue` must only appears in a loop (not a switch this time)
- After `continue`, the condition of the loop is checked first.
- Exit the loop if the condition is false.

continue Statement

```java
int age;
String name = "";
do {
   System.out.print("Enter your age:\t");
   age = scanner.nextInt();
   if (age < 12) {
       System.out.println("You are not old enough");
       continue;
   }
   System.out.print("Enter your name:");
   name = scanner.next();
} while (age < 12);
System.out.println("Welcome " + name);
```

```
Enter your age: 10
You are not old enough
Enter your age: 15
Enter your name:Kevin
Welcome Kevin
```

break and continue

- Both `break` and `continue` skips the remaining loop body.
- `break` exit the loop regardless the condition.
- `continue` exit the loop only if the condition is false, repeat otherwise.
- `continue` cannot be used in a switch (it is not a loop!)

Quick Summary

- while loop
- do while loop
- break
- continue

For Loop

for loop

- `for`-loop executes a predetermined number of times
- Similar to while, loop body should be embraced by curly bracket `{}`.

```
for (<init-expr>; <condition>; <post-iter-action>) {
    <loop body>
}
```

Example

```
for (int i = 0; i < 10; i++)
    System.out.print(i + " ");    // {} is optional for one statement
```

```
0 1 2 3 4 5 6 7 8 9
```

for loop

```
for (<init-expr>; <condition>; <post-iter-action>) {
    <loop body>
}
```

<init-expr>

- to declare and initial a counter variable
- executed only once at the start of loop

<condition>

- a boolean. Repeat if true
- evaluated before each iteration, including the first time (like `while`)

<post-iter-action>

- action that is taken after each iteration
- usually used for update the counter variable

for loop

```
for (int i = 0; i < 10; i++) {
    System.out.print(i + " ");
}
```

<init-expr>

① `int i = 0;`

- executed only once at the start of loop

<condition>

② `i < 10`

- evaluated before each iteration, including the first time (like `while`)

<post-iter-action>

③ `i++`

- action that is taken after each iteration

Execution sequence:

Enter loop ➡️ ①②🅱③②🅱③②🅱③②🅱③②🅱③...②🅱③ ➡️ Exit loop

for loop - example 2

```java
for (int i = 0; i < 5; i += 2)
    System.out.println("i : " + i);
```

```
i : 0
i : 2
i : 4
```

Sequence of execution

Enter loop ➡ ①②🅱③②🅱③②🅱③② ➡ Exit loop

- ①: `int i = 0;`
- ②: `i < 5`
- ③: `i+= 2`

for loop - number of iterations

How many times the following loops are executing?

```
for (int i = 0; i < 60; i++)     //common practice
    System.out.println("Hello");
```

```
for (int i = 1; i < 60; i++)
    System.out.println("Hello");
```

```
for (int i = 0; i <= 60; i++)
    System.out.println("Hello");
```

```
for (int i = 1; i <= 60; i++)
    System.out.println("Hello");
```

for loop

- The variable declared in <init-expr> has a *scope* within the loop only, i.e., not visible after the loop.

```
for (int i = 0; i < 5; i += 2)
    System.out.println("i : " + i);
System.out.println(i); //Error!
```

- If the counter variable is still useful after the loop, declare it outside the loop!

```
int i = 0;
for (; i < 5; i += 2)
    System.out.println("i : " + i);
System.out.println(i); //OK
```

```
i : 0
i : 2
i : 4
6
```

Empty expressions in for loop

- <init-expr>, <condition>, <post-iter-action> are **optional** in for loop

```
int i;
for (; i < 4; i++) {}  //omit <init-expr>
for (int j ; i < 4;) { i++; } //omit <post-iter-action>
for ( ; ; ) { i++; } //omit condition
```

- The expression `for (;;)` is same as `while (true)`
- Remember to put a break when using `for (;;)`

Multiple expressions in for loop

- <init-expr>: can define multiple variables of the same type separated by `,`

```
for (int i = 0, j; i < 5; i++) { ... }
```

- <condition>: can define more complicate logic using `&&` `||`

```
for (int i = 0; i < 3 && input != password; i++) { ... }
```

- <post-iter-action>: can define multiple actions separated by `,`

```
for (int i = 0, j = 10; i < 10; i++, j--) { ... }
```

🧑‍🏫 💬 Avoid multiple expressions <init-expr> and <post-iter-action> if you are not familiar with the language.

Enhanced for loop

- Used to traverse an array and *enumerable*.
- Sometimes also known as **for-each** loop.
- Will revisit again when we have array and enumerable like `Vector`, `List`.

```
for (<declaration> : <array or enumerable>) {
    <loop body>
}
```

Example

```
int[] array = {2,0,2,6};
for (int i : array) {
    System.out.println("i: " + i);
}
```

```
i: 2
i: 0
i: 2
i: 6
```

Enhanced for loop

```
for (<declaration> : <array or enumerable>) {
    <loop body>
}
```

<declaration>

- Declare exactly **ONE variable**.
- Type compatible with the elements of the array/enumerable.
- The scope of the variable is inside the loop only.

<array or enumerable>

- The array or enumerable you need to loop through

Remarks about Enhanced for loop

- It is useful when you need to process all elements of the array
- Can be used for stepping through elements of an array in first-to-last order
- Simple but less flexible
- The variable is copied by value not by reference
- **Do not know** the index of the current element

👨‍🏫 💬 You may find this page difficult to understand when you first read this at the beginning of the semester. You will (hopefully) understand that a few weeks after. This page will not be part of the first quiz.

Same task, different loops

- Sometimes you can express the same logic using the different loops

Example 1

```java
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

```java
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
```

Example 2 - find largest factor other than itself (for num >= 2)

```java
int i, num = 103;
for (i = num - 1; i > 1; i--)
    if (num % i == 0)
        break;
System.out.print("factor: " + i);
```

```java
int num = 103, i = num;
do {
    i--;
} while (i > 1 && num % i != 0);
System.out.print("factor: " + i);
```

Same task, different loops

```java
Scanner s = new Scanner(System.in);
int age = 0;
do {
    age = s.nextInt();
    if (age < 12)
        System.out.println("Too young!");
} while (age < 12);
System.out.println("Age = " + age);
```

```java
Scanner s = new Scanner(System.in);
int age = s.nextInt();
while (age < 12) {
    if (age < 12)
        System.out.println("Too young!");
    age = s.nextInt();
}
System.out.println("Age = " + age);
```

```java
Scanner s = new Scanner(System.in);
int age;
while (true) {
    age = s.nextInt();
    if (age >= 12)
        break;
    System.out.println("Too young!");
}
System.out.println("Age = " + age);
```

```java
Scanner s = new Scanner(System.in);
int age;
boolean valid = false;
while (!valid) {
    age = s.nextInt();
    valid = (age >= 12);
    if (!valid)
        System.out.println("Too young!");
}
System.out.println("Age = " + age);
```

How to choose the right loop

- There is no absolute answer

for loop

- usually used with a known number of iteration; or
- when the number of iteration executed is important;

while loop / do-while loop

- usually used with an indefinite number of iteration

> 🧑‍🏫 💬 Use the **cleanest** and **simplest** one if more than one type of loop can solve the problem.

Summary

- for-loop
- Enhanced for-loop

Nested Loop

Printing Square

- We want to build a program that prints a square of size input from the user.

```
input the size of the rectangle: 5
*****
*****
*****
*****
*****
```

Printing Square - A very bad answer

```
Scanner s = new Scanner(System.in);
int size = s.nextInt();
switch (size) {
    case 1: System.out.println("*"); break;
    case 2: System.out.println("**");
            System.out.println("**");break;
    case 3: System.out.println("***");
            System.out.println("***");
            System.out.println("***"); break;
    default: System.out.println("I give up");
}
```

🧑‍🏫 💬 Why this is a bad answer?

Printing Square - Solution

- How many rows we need to print?

```
for (int i = 0; i < size; i++) {
    //print enough stars
}
```

- On each row, how many stars we need to print?

```
for (int i = 0; i < size; i++)
    System.out.print("*");
```

- Now try to combine the two loops together into a **nested loop**.

Printing Square - Solution

```java
for (int i = 0; i < size; i ++) {
    //print enough stars
    //remember, can't use the same variable name
    for (int j = 0; j < size; j++)
        System.out.print("*");
}
```

```
input the size of the rectangle: 3
*********
```

- Almost there, we need to hit an `enter` per each row.

Printing Square - Complete Solution

```
Scanner s = new Scanner(System.in);
int size = s.nextInt();
for (int i = 0; i < size; i ++) {
    //print enough stars
    //remember, can't use the same variable name
    for (int j = 0; j < size; j++)
        System.out.print("*");

    //Hit an enter key
    System.out.println();
}
```

```
input the size of the rectangle: 3
***
***
***
```

Let's see how good is your math

- What is the answer for $x = 1 + 2 + 3 + \ldots + 100$?
- How about $y = 1 + 3 + 5 + 7 + \ldots + 101$?
- How about $z = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 + \ldots + 100^2$?
- How about $w = 1^2 + 3^2 + 5^2 + 7^2 + \ldots + 101^2$
- CS students got a CS solution!

Solution

$$x = 1 + 2 + 3 + \ldots + 100$$
$$y = 1 + 3 + 5 + 7 + \ldots + 101$$
$$z = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 + \ldots + 100^2$$
$$w = 1^2 + 3^2 + 5^2 + 7^2 + \ldots + 101^2$$

```
int x, y, z, w;
x = y = z = w = 0;
for (int i = 1; i <= 100; i++)
    x += i;
for (int i = 1; i <= 101; i += 2)
    y += i;
for (int i = 1; i <= 100; i++)
    z += i * i;
for (int i = 1; i <= 101; i += 2)
    w += i * i;
```

How about this now..

$t = 1! + 2! + 3! + 4! + .. + 10!$, such that, $i! = 1 \times 2 \times 3.. \times i, \forall i \geq 1$

- If I know what is the value of each $i!$, I can sum them.
- And If I know the value of i, I can compute $i!$ too.

Rough Idea

```
int t = 0;
for (int i = 1; i <= 10; i++)
    t += i_factorial; //only if I know what is i_factorial
```

```
int i_factorial = 1;
for (int j = 1; j <= i; j++)
    i_factorial *= j; //if i is given
```

Now combine them together

$t = 1! + 2! + 3! + 4! + .. + 10!$, such that, $i! = 1 \times 2 \times 3.. \times i, \forall i \geq 1$

```
int t = 0;
for (int i = 1; i <= 10; i++) {
    //inner logic to compute i_factorial
    int i_factorial = 1;
    for (int j = 1; j <= i; j++)
        i_factorial *= j;

    System.out.println(i + "! = " + i_factorial);
    t += i_factorial;
}
```

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
t: 4037913
```

Deeper look at nested loop

```java
int t = 0;
for (int i = 1; i <= 10; i++) {
    //inner logic to compute i_factorial
    int i_factorial = 1;
    for (int j = 1; j <= i; j++)
        i_factorial *= j;

    System.out.println(i + "! = " + i_factorial);
    t += i_factorial;
}
```

i	j	Remark
1	1	end j-loop
2	1	
2	2	end j-loop
3	1	
3	2	
3	3	end j-loop
4	1	
4	2	
4	3	
4	4	end j-loop
..	..	

Nested Loop

- Theoretically speaking, you can have many many layers of loops
- Just make sure you need them all because the total number of iterations are **multiplying**!
- Each layer should have different variable names.

```
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        int k = 0;
        while (k < 10) {
            k += i + j;
        }
        System.out.print(k);
    }

    //j can reuse here since the previous j-loop ended.
    for (int j = 0; j < i; j++)
        System.out.print(j);
}
```

Break/Continue inside Nested Loop

- `break` and `continue` inside a nested loop makes an effect on immediate parent only.

```
System.out.println("i j k");
for (int i = 0; i < 2; i++)
    for (int j = 0; j < 2; j++)
        for (int k = 0; k < 3; k++) {
            System.out.printf("%d %d %d ", i, j, k);
            if (k < j) {
                System.out.println("continue");
                continue;
            }
            if (k > i + j) {
                System.out.println("break");
                break;
            }
            System.out.println("*");
        }
```

```
i j k
0 0 0 *
0 0 1 break
0 1 0 continue
0 1 1 *
0 1 2 break
1 0 0 *
1 0 1 *
1 0 2 break
1 1 0 continue
1 1 1 *
1 1 2 *
```

Summary

- while loop
- do-while loop
- for loop
- enhanced for loop
- continue
- break
- Nested Loop