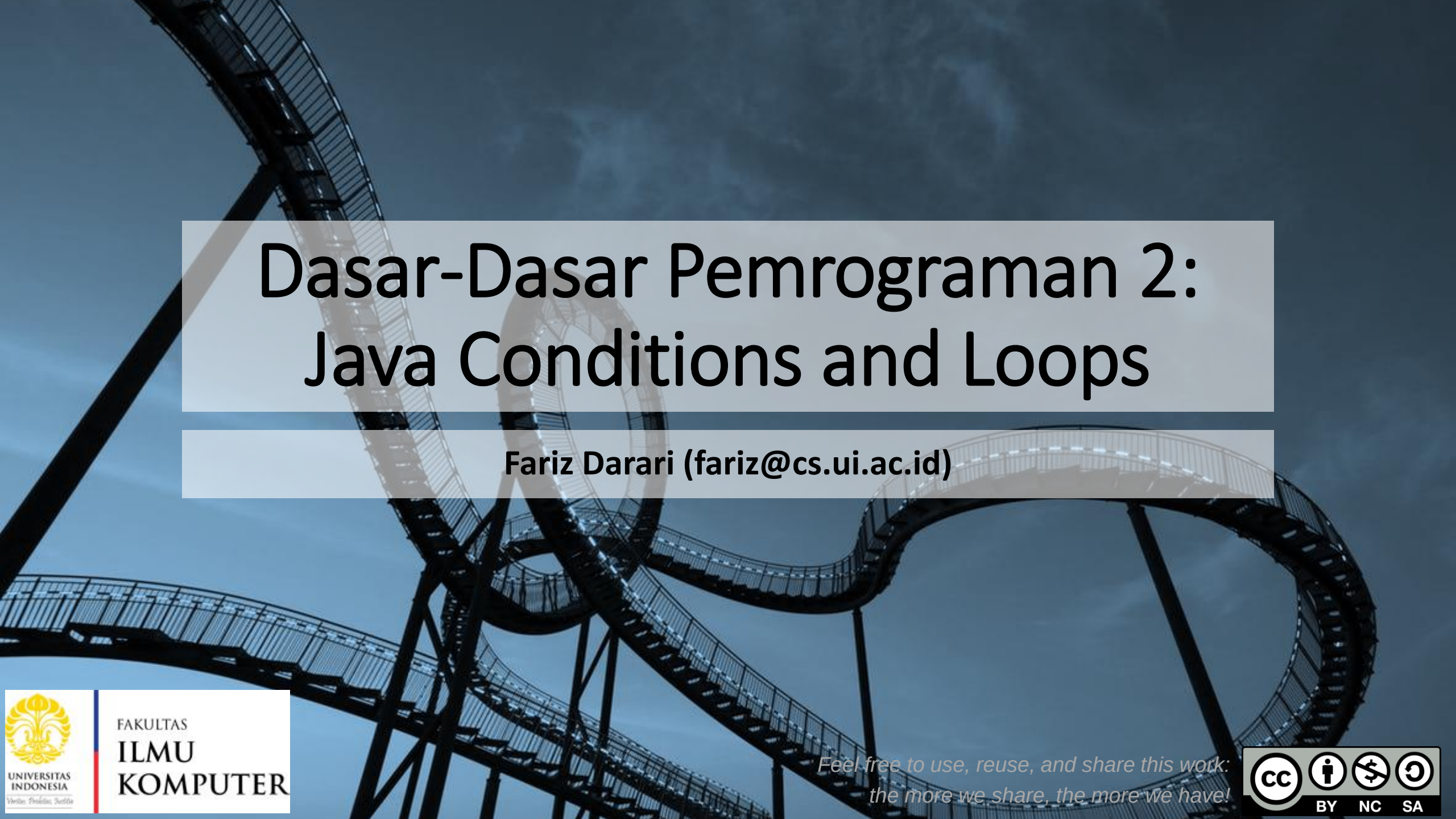

Dasar-Dasar Pemrograman 2: Java Conditions and Loops

Fariz Darari (fariz@cs.ui.ac.id)

UNIVERSITAS INDONESIA

FAKULTAS ILMU KOMPUTER

Relational operators

Relational operators are used to check conditions like whether two values are equal, or whether one is greater than the other. The following expressions show how they are used:

```
x == y          // x is equal to y
x != y          // x is not equal to y
x > y           // x is greater than y
x < y           // x is less than y
x >= y          // x is greater than or equal to y
x <= y          // x is less than or equal to y
```

The result of a relational operator is one of two special values, `true` or `false`. These values belong to the data type `boolean`; in fact, they are the only boolean values.

Java has three logical operators

&& that is used for **and**

|| that is used for **or**

! that is used for **not**

Examples

For example, `x > 0 && x < 10` is true when `x` is both greater than zero *and* less than 10. The expression `evenFlag || n % 3 == 0` is true if either condition is true, that is, if `evenFlag` is true *or* the number `n` is divisible by 3. Finally, the `!` operator inverts a boolean expression. So `!evenFlag` is true if `evenFlag` is *not* true.

De Morgan's laws

If you ever have to negate an expression that contains logical operators, and you probably will, **De Morgan's laws** can help:

- `!(A && B)` is the same as `!A || !B`
- `!(A || B)` is the same as `!A && !B`

Conditions

To write useful programs, we almost always need to check conditions and react accordingly. **Conditional statements** give us this ability. The simplest conditional statement in Java is the `if` statement:

```
if (x > 0) {
    System.out.println("x is positive");
}
```

The expression in parentheses is called the condition. If it is true, the statements in braces get executed. If the condition is false, execution skips over that block of code. The condition in parentheses can be any boolean expression.

if and else

A second form of conditional statement has two possibilities, indicated by `if` and `else`. The possibilities are called **branches**, and the condition determines which one gets executed:

```
if (x % 2 == 0) {
    System.out.println("x is even");
} else {
    System.out.println("x is odd");
}
```

If the remainder when `x` is divided by 2 is zero, we know that `x` is even, and this fragment displays a message to that effect. If the condition is false, the second print statement is executed instead. Since the condition must be true or false, exactly one of the branches will run.

Quiz time: What's wrong?

```
if (x > 0)
    System.out.println("x is positive");
    System.out.println("x is not zero");
```

Quiz time: What's wrong?

```
if (x > 0)
    System.out.println("x is positive");
    System.out.println("x is not zero");
```

This code is misleading because it's not indented correctly. Since there are no braces, only the first `println` is part of the `if` statement. Here is what the compiler actually sees:

```
if (x > 0) {
    System.out.println("x is positive");
}
    System.out.println("x is not zero");
```

As a result, the second `println` runs no matter what. Even experienced programmers make this mistake; search the web for Apple's "goto fail" bug.

Chaining if and else

Sometimes you want to check related conditions and choose one of several actions. One way to do this is by **chaining** a series of `if` and `else` statements:

```
if (x > 0) {
    System.out.println("x is positive");
} else if (x < 0) {
    System.out.println("x is negative");
} else {
    System.out.println("x is zero");
}
```

These chains can be as long as you want, although they can be difficult to read if they get out of hand. One way to make them easier to read is to use standard indentation, as demonstrated in these examples. If you keep all the statements and braces lined up, you are less likely to make syntax errors.

Nesting if and else

In addition to chaining, you can also make complex decisions by **nesting** one conditional statement inside another. We could have written the previous example as:

```
if (x == 0) {
    System.out.println("x is zero");
} else {
    if (x > 0) {
        System.out.println("x is positive");
    } else {
        System.out.println("x is negative");
    }
}
```

while loop

```
public static void countdown(int n) {
    while (n > 0) {
        System.out.println(n);
        n = n - 1;
    }
    System.out.println("Blastoff!");
}
```

You can almost read the `while` statement like English: "While `n` is greater than zero, print the value of `n` and then reduce the value of `n` by 1. When you get to zero, print Blastoff!"

while loop

The expression in parentheses is called the condition. The statements in braces are called the **body**. The flow of execution for a `while` statement is:

1. Evaluate the condition, yielding `true` or `false`.
2. If the condition is `false`, skip the body and go to the next statement.
3. If the condition is `true`, execute the body and go back to step 1.

This type of flow is called a **loop**, because the last step loops back around to the first.

for loop

```
public static void countdown(int n) {

    for(int i = n; i > 0; i--) {
        System.out.println(i);
    }
    System.out.println("Blastoff!");

}
```

for loop

`for` loops have three components in parentheses, separated by semicolons: the initializer, the condition, and the update.

1. The *initializer* runs once at the very beginning of the loop.
2. The *condition* is checked each time through the loop. If it is `false`, the loop ends. Otherwise, the body of the loop is executed (again).
3. At the end of each iteration, the *update* runs, and we go back to step 2.

do-while loop

The `while` and `for` statements are **pretest loops**; that is, they test the condition first and at the beginning of each pass through the loop.

Java also provides a **posttest loop**: the `do-while` statement. This type of loop is useful when you need to run the body of the loop at least once.

```
int i = 0;
do {
     System.out.println("Hi!");
} while (i!=0);
```

Try run this, check if Hi! is still printed out or not

take a break

```
int i = 0;
while(i < 6) {
     System.out.print(i);
     if(i == 3)
          break;
     i++;
}
```

0123

Check this out too: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html

switch to be a better person

```
String you = "good";
switch(you) {
     case "bad":
          you = "good";
          break;
     case "good":
          you = "better";
          break;
     default:
          you = "best";
}
System.out.println(you);
```

Credits: Think Java book by Allen Downey and Chris Mayfield