Programming Fundamentals II

Conditional Control Structures

Let’s recap the previous session

```
class NameAndAge {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    System.out.print("Name: ");
    String name = sc.nextLine();

    System.out.print("Age: ");
    int age = sc.nextInt();

    System.out.println(name + " is " + age + " years old.");
  }
}
```

We’ve discussed…

- Simple Java program with a class and a `main` method
- How to write, compile, and run a Java program
- Variables, values, and expressions
- Primitive and reference types
- Widening and narrowing type conversion
- Naming rules and conventions
- Basic I/O and message dialogs
- Code of ethics

Let’s continue our journey…

The two forms of if

```
if (condition)
      true-branch
```

```
if (condition)
      true-branch
else
      false-branch
```

```
int x = 10;
int y = 20;
boolean b = true;
boolean c = b && x < y;
```

```
if (x < y - 10)
    System.out.println("First case is true");
```

if without else

```
if (c) {
    System.out.println("Second case is true");
    System.out.println("And I like it!");
} else {
    System.out.println("Second case is false");
    System.out.println("Which I don’t like");
}
```

if-else with block statements

What is a block statement?

- A block statement can appear anywhere where a statement is expected
- Used for grouping multiple statements into one
- A block statement defines a scope of variables declared inside the block
    - Variables declared inside the block are not visible outside the block
    - Their lifetimes also begin and end inside the block

```
int a = 10;
double b = 5.25;

System.out.println("Outside the block");

{
    double c = 15.4;

    System.out.println("Inside the block");
    System.out.println("... where a + c = " + (a + c));
}

System.out.println("Outside again");
System.out.println("... where a + b = " + (a + b));
System.out.println("Referring c here will cause an error");
```

This part of code does not see the variable c

This doesn’t work without block statements

```
int x = 10;
int y = 20;
boolean b = true;
boolean c = b && x < y;

if (x < y - 10)
    System.out.println("First case is true");

if (c)
    System.out.println("Second case is true");
    System.out.println("And I like it!");
else
    System.out.println("Second case is false");
    System.out.println("Which I don’t like");
```

This line will cause a syntax error

This doesn’t work either

```
int x = 10;
int y = 20;
boolean b = true;
boolean c = b && x < y;

if (x < y - 10)
    System.out.println("First case is true");

if (c)
    System.out.println("Second case is true");
else
    System.out.println("Second case is false");
    System.out.println("Which I don’t like");
```

This line will be executed regardless of the condition

We’ve seen Boolean expressions in action

```
int x = 10;
int y = 20;
boolean b = true;
boolean c = b && x < y;
```

- Boolean expressions are expressions that yield a value of either `true` or `false`

Boolean operators yield Boolean results

- Comparison operators
  - <, >, <=, >=, ==, !=

- Logical operators
  - !, &&, ||

- Only a Boolean expression can be used as the condition of an if statement

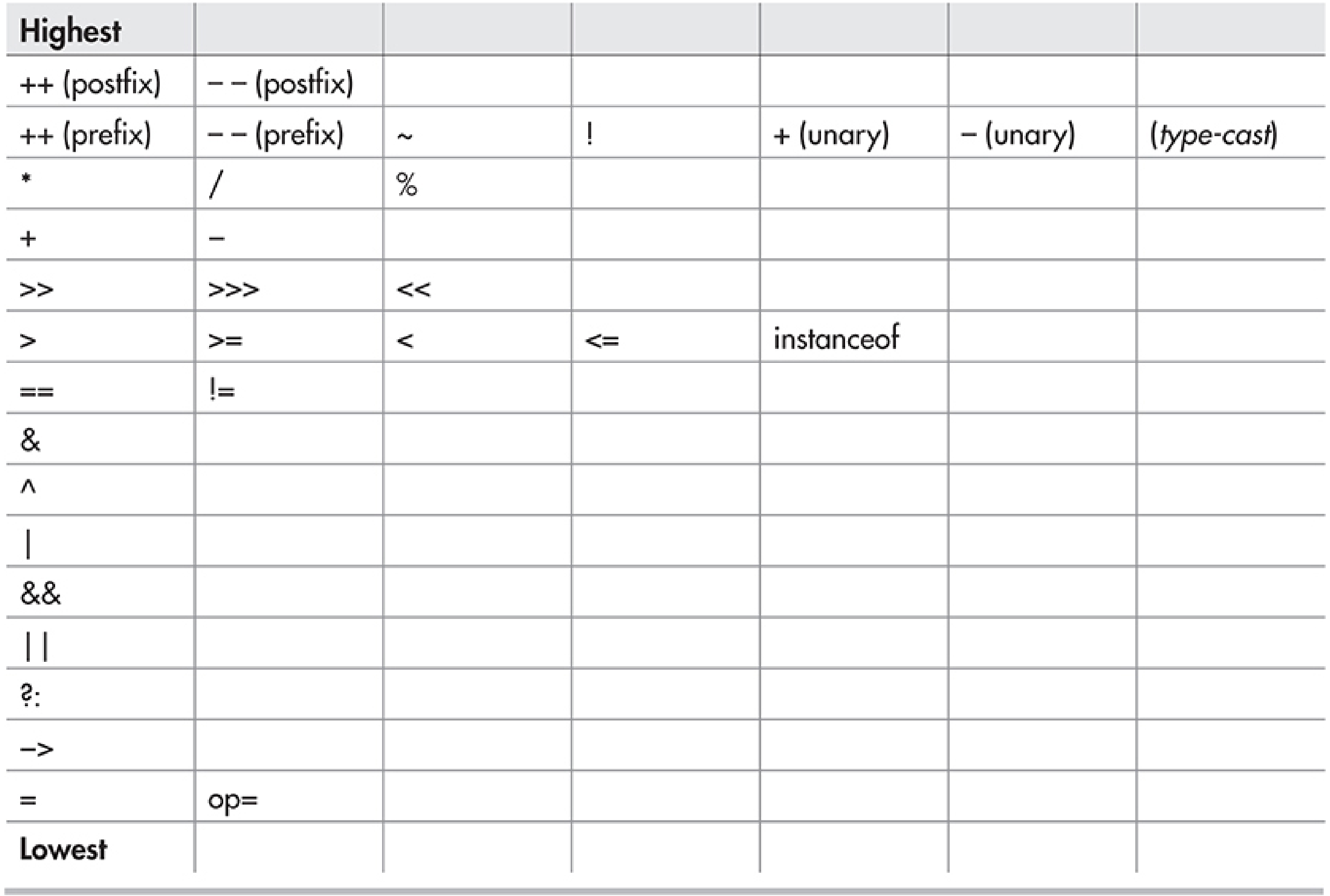
Are the followings Boolean expressions?

Given

```
int x = 10; int y = 20;
boolean b = true; boolean c = b && x < y;
```

Expression	**Boolean?**	**Value**
`c`	○	`true`
`x >= y`	○	`false`
`x + y == 40`	○	`false`
`x <= 10 || x > 60 && y < 20`	○	`true`
`b || c`	○	`true`

Operator Precedence



Highest						
++ (postfix)	– – (postfix)					
++ (prefix)	– – (prefix)	~	!	+ (unary)	– (unary)	(*type-cast*)
*	/	%				
+	–					
>>	>>>	<<				
>	>=	<	<=	instanceof		
==	!=					
&						
^						
|						
&&						
||						
?:						
–>						
=	op=					
Lowest						

Table from Java: A Beginner’s Guide

Short-circuit evaluation: which parts?

```
(a < b || c < d) && !(a < d)

int a = 1, b = 2, c = 3, d = 4;

(a < b || c < d) && !(a < d)

int a = 4, b = 3, c = 2, d = 1;

(a < b || c < d) && !(a < d)

int a = 5, b = 4, c = 4, d = 5;

(a < b || c < d) && !(a < d)
```

There is a strict version of these operators

- These operators always evaluate both operands
  - They never short-circuit
- & (strict AND), | (strict OR), and ^ (strict XOR)
- But they are rarely used
  - We will stick with the normal version for the rest of this course

What’s the result of this code?

```
double c = 0.1;
double d = 0.2;
if (c + d == 0.3)
    System.out.println("Yay!!!");
else
    System.out.println("Nay...");
```

```
Nay...
```

Be careful with floating-point comparison

- What is the result of 0.1 + 0.2?
- Whatever it should be, Java says it is 0.30000000000000004

Dealing with floating-point imprecision

```
double c = 0.1;
double d = 0.2;

// Error threshold that is acceptable to you
double tolerance = 0.000001;

if (Math.abs(c + d - 0.3) < tolerance)
    System.out.println("Yay!!!");
else
    System.out.println("Nay...");
```

Compared to:

`(c + d == 0.3)`

Conditional expressions can shorten your code

```
int x = -10;
int y = x < 0 ? -x : x;
System.out.println(y);
```

Result:

```
10
```

Syntax:

condition ? *value*$_1$: *value*$_2$

The if-else-if ladder

```
int score = 75;
String grade;

if (score < 50)
    grade = "F";
else if (score < 60)
    grade = "D";
else if (score < 70)
    grade = "C";
else if (score < 80)
    grade = "B";
else
    grade = "A";

System.out.println(grade);
```

The switch statement

```
switch ($expression$) {
    case $constant_1$:
        $branch_1$
    case $constant_2$:
        $branch_2$
    ...
    default:
        $branch_n$
}
```

This part is optional

```
int day = 4;
String dayName;

switch (day) {
    case 1:
        dayName = "Sunday";
        break;
    case 2:
        dayName = "Monday";
        break;
    case 3:
        dayName = "Tuesday";
        break;
    case 4:
        dayName = "Wednesday";
        break;
    case 5:
        dayName = "Thursday";
        break;
    case 6:
        dayName = "Friday";
        break;
    case 7:
        dayName = "Saturday";
        break;
    default:
        dayName = "No such day";
}

System.out.println(dayName);
```

- Use `switch` when conditions are of constant values
  - Otherwise, use `if`

- `switch` is usually used with `break`s in its `case`s

- Case constants must be one of the following:
  - Integer types: `int`, `short`, `char`, `byte`
  - A boxed version of the above (discussed in the next lesson)
  - An `enum` type
  - `String`

Without breaks, switch falls through

- Always add a `break` at the end of every case
  - Unless you really want the fall-through behavior

- Possible exceptions:
  - You want to group multiple case labels
  - You want some case actions to also include actions of following cases

```
Random dice = new Random();
String prizes = "You've got ";
switch (dice.nextInt(6) + 1) {
    case 1:
        prizes += "a teddy bear.";
        break;
    case 2:
        prizes += "a model robot, ";
    case 3:
        prizes += "a board game, ";
    case 4: case 5:
        prizes += "a lollipop, ";
    case 6:
        prizes += "a fancy mask, ";
    default:
        prizes += "and a lot of fun.";
}
System.out.println(prizes);
```

What’s the result of the program if the dice rolls 1?

What about 2? 5? Or 6?

The while loop

```
while (condition)
     body
```

- Check, then act

Here’s an example, but what’s a ++ (or --)?

```
int x = 10;
int y = 5;
int sum = 0;

while (x > y) {
    sum += x + y;
    x--;
    y++;
}

System.out.println("Total: " + sum);
```

Compound assignments (+=, -=, *=, /=, etc.) are basically the same as in Python

Pre- and post-increment and decrement are discussed on the next page

Pre- and post-increment and decrement

```
int x = 10;
int y = ++x + 5;
System.out.println("x, y = " + x + ", " + y);
```

```
x, y = 11, 16
```

```
int x = 10;
int y = x++ + 5;
System.out.println("x, y = " + x + ", " + y);
```

```
x, y = 11, 15
```

What’s the result of this code?

```
int n = 5;
int m = ++n;
double x = 2.5;
double y = x++;

System.out.println("n="+n+" m="+m+" x="+x+" y="+y);
```

```
n=6 m=6 x=3.5 y=2.5
```

```
y = n++ + --m + x--;

System.out.println("n="+n+" m="+m+" x="+x+" y="+y);
```

```
n=7 m=5 x=2.5 y=14.5
```

And what’s the result of this code?

```java
int x = 10;
int y = 5;
int sum = 0;

while (x > y) {
    sum += x + y;
    x--;
    y++;
}

System.out.println("Total: " + sum);
```

```
Total: 45
```

The do … while loop

```
do
        body
while (condition);
```

- Act, then check

```java
int secret = random.nextInt(100) + 1;
int guess;
int count = 0;
int limit = 10;
do {
  System.out.println("Enter your guess (1-100): ");
  guess = sc.nextInt();
  count++;
  if (guess > secret)
    System.out.println("Too high. Try again.");
  else if (guess < secret)
    System.out.println("Too low. Try again.");
  else
    System.out.println("You won! Total: " + count + " tries.");
} while (guess != secret && count < limit);

if (count == limit)
  System.out.println("You've exceeded guess limit.");
```

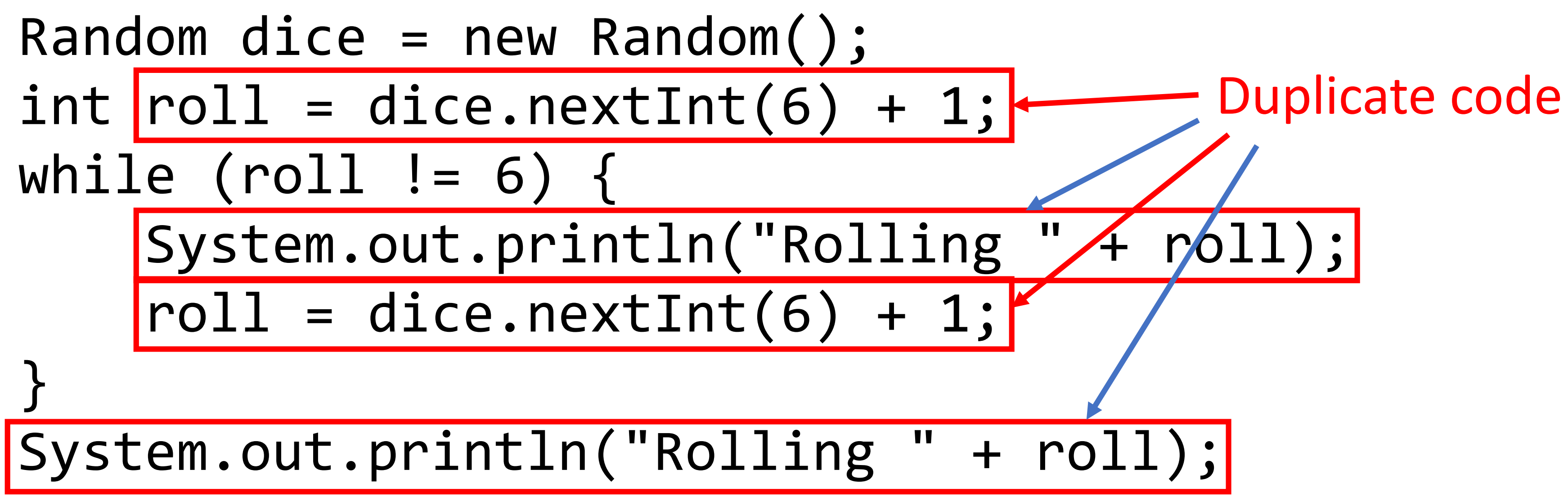
```
Enter your guess (1-100):
50
Too low. Try again.
Enter your guess (1-100):
75
Too high. Try again.
Enter your guess (1-100):
62
Too low. Try again.
Enter your guess (1-100):
68
Too low. Try again.
Enter your guess (1-100):
71
Too low. Try again.
Enter your guess (1-100):
73
You won! You've guessed 6 times.
```

```
int secret = random.nextInt(100) + 1;
int guess;
int count = 0;
int limit = 10;
do {
  System.out.println("Enter your guess.
  guess = sc.nextInt();
  count++;
  if (guess > secret)
    System.out.println("Too high. Try..
  else if (guess < secret)
    System.out.println("Too low. Try...
  else
    System.out.println("You won! Total.
} while (guess != secret && count < lim

if (count == limit)
  System.out.println("You've exceeded..
```

When to use which?

- `while` is more commonly used and suitable in most cases
- Use do … `while` when one of the followings holds:
  - You need to always have at least one iteration regardless of the condition
  - You have an action *p* that needs to be done on every iteration, and the condition *c* depends on the result of action *p*
  - In this case, if you use `while`, you will need to duplicate *p* both before the `while` clause and within the body of the `while`, so do … `while` is a better fit



```
Random dice = new Random();
int roll = dice.nextInt(6) + 1;
while (roll != 6) {
    System.out.println("Rolling " + roll);
    roll = dice.nextInt(6) + 1;
}
System.out.println("Rolling " + roll);
```

Duplicate code

Result:

```
Rolling 2
Rolling 3
Rolling 1
Rolling 6
```

```
Random dice = new Random();
int roll;
do {
    roll = dice.nextInt(6) + 1;
    System.out.println("Rolling " + roll);
} while (roll != 6);
```

This loop always executes at least once even if roll is 6 `break` and `continue` work the same way as in Python

```
// Sum all positive integers less than 100, skipping those that
// has 7 as a factor, until the sum exceeds 500

int sum = 0;
int count = 1;

while (count < 100) {
    if (count % 7 == 0) {
        count++;
        continue;
    }
    sum += count;
    if (sum > 500)
        break;
    count++;
}

System.out.println("Stopped at " + count + " with the sum " + sum);
```

Result:

```
Stopped at 34 with the sum 525
```

Today, we've discussed…

- Boolean expressions
- Short-circuit evaluation
- Floating-point comparison
- Conditional expressions

- Block statements
- Block scope

- `if`/`if-else`
- `if-else-if` ladder

- `switch`
- Multiple labels and fall-through

- `while`
- `do-while`
- `break` and `continue`

- Pre- and post-increment and decrement
- Compound assignment

Related lecture notes

- 02 – Control Structures
  - Link: https://goo.gl/jF5a9Y