FCDS

Programming I

Lecture 4: boolean, Logical Expressions, Conditional Execution

Java's primitive types

- **primitive types**: there are 8 simple types for numbers, text, etc.

Type	Description	Size
`int`	The integer type, with range -2,147,483,648 . . . 2,147,483,647	4 bytes
`byte`	The type describing a single byte, with range -128 . . . 127	1 byte
`short`	The short integer type, with range -32768 . . . 32767	2 bytes
`long`	The long integer type, with range -9,223,372,036,854,775,808 . . . -9,223,372,036,854,775,807	8 bytes
`double`	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
`float`	The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes
`char`	The character type, representing code units in the Unicode encoding scheme	2 bytes
`boolean`	The type with the two truth values `false` and `true`	1 byte

The ***boolean*** data type

Useful to control flow and logic in programs (see later).

values	true or false		
literals	`true` `false`		
operations	and	or	not
operators	&&	||	!

Boolean Operators

a	!a
true	false
false	true

a	b	a && b	a || b
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

Truth-table definitions of `boolean` *operations*

Relational Operators

op	*meaning*	`true`	`false`
`==`	*equal*	`2 == 2`	`2 == 3`
`!=`	*not equal*	`3 != 2`	`2 != 2`
`<`	*less than*	`2 < 13`	`2 < 2`
`<=`	*less than or equal*	`2 <= 2`	`3 <= 2`
`>`	*greater than*	`13 > 2`	`2 > 13`
`>=`	*greater than or equal*	`3 >= 2`	`2 >= 3`

non-negative discriminant?	`(b*b - 4.0*a*c) >= 0.0`
beginning of a century?	`(year % 100) == 0`
legal month?	`(month >= 1) && (month <= 12)`

Java Operator Precedence

Table 4.2 Java Operator Precedence

Description	Operators
unary operators	++, --, +, -
multiplicative operators	*, /, %
additive operators	+, -
relational operators	<, >, <=, >=
equality operators	==, !=
assignment operators	=, +=, -=, *=, /=, %=

Boolean Operators

Not !

And &&

Or ||

Example

Example	Result
`(2 == 3) && (-1 < 5)`	`false`
`(2 == 3) || (-1 < 5)`	`true`
`!(2 == 3)`	`true`

Evaluating *logic expressions*

- Relational operators have lower precedence than math.

```
5 * 7 >= 3 + 5 * (7 - 1)
5 * 7 >= 3 + 5 * 6
35    >= 3 + 30
35    >= 33
true
```

- Relational operators cannot be "chained" as in algebra.

```
2 <= x <= 10
true   <= 10
error!
```

(assume that `x` is `15`)

  – Instead, combine multiple tests with **&&** or **||**

```
2 <= x && x <= 10
true   && false
false
```

Logical questions

- What is the result of each of the following logical expressions?

```
int x = 42;
int y = 17;
int z = 25;
```

  - `y < x && y <= z`
  - `x % 2 == y % 2 || x % 2 == z % 2`
  - `x <= y + z && x >= y + z`
  - `!(x < y && x < z)`
  - `(x + y) % 2 == 0 || !((z - y) % 2 == 0)`

    - Answers: `true`, `false`, `true`, `true`, `false`

Example

```
public static void main(String[] args) {
    boolean b1 = true;
    boolean b2 = false || b1;

    System.out.println(b2);

    int x = 3;
    int y = 4;

    b1 = (x == y) || (x <= y);
    b2 = x > y;

    boolean b3 = (x >= 1) && (x <= 5);

    System.out.println(b1 + " " + b2 + " " + b3);
}
```

```
true
true false true
```

Conditional Execution

The following ***Selection Statements*** are used to control the flow of the program in Java:

- if statement
- if-else statement
- switch statement

The `if` statement

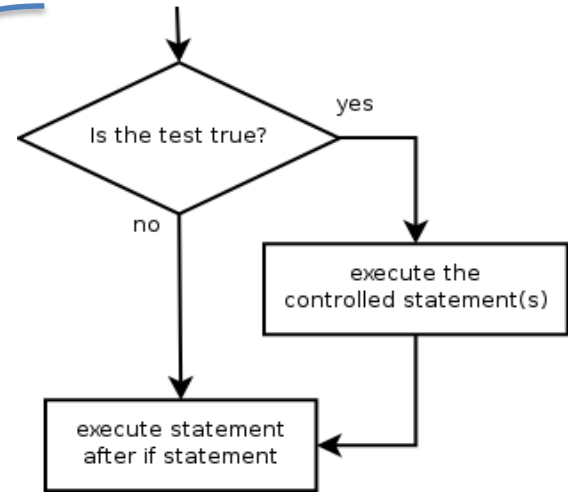
Executes a block of statements only if a test is true

```
if (test) {
    statement;
    ...
    statement;
}
```

Logical Expression

Controlled Statements

Flow Chart



- Example:

```
double gpa = console.nextDouble();
if (gpa >= 2.0) {
    System.out.println("Application accepted.");
}
```

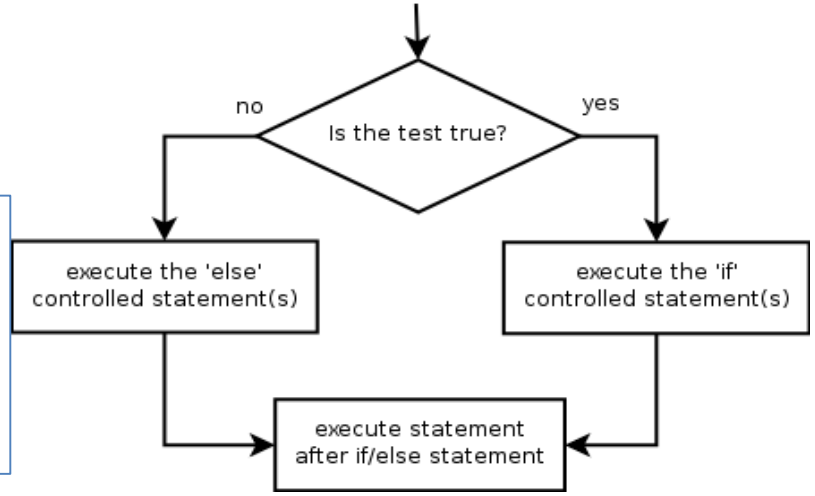
The `if/else` statement

Executes one block if a test is true, another if false

```
if (test) {
    statement(s);
} else {
    statement(s);
}
```

if controlled statements (body of if)

else controlled statements (body of else)



- Example:

```
double gpa = console.nextDouble();
if (gpa >= 2.0) {
    System.out.println("Welcome to BAU!");
} else {
    System.out.println("Application denied.");
}
```

Statement Types

- The body of `if` statement may consist of multiple statements that must be executed in sequence whenever the condition is true.
  - These statements must be grouped together to form a *block statement* by enclosing them in braces { }

```
if (amount <= balance)
{
   double newBalance = balance - amount;
   balance = newBalance;
 }
```

- Different types of statements:
  - Simple statement:

```
balance = balance - amount;
```

  - Compound statement :

```
if (balance >= amount) balance = balance - amount;
```

  - Block statement

```
{
    double newBalance = balance - amount;
    balance = newBalance;
 }
```

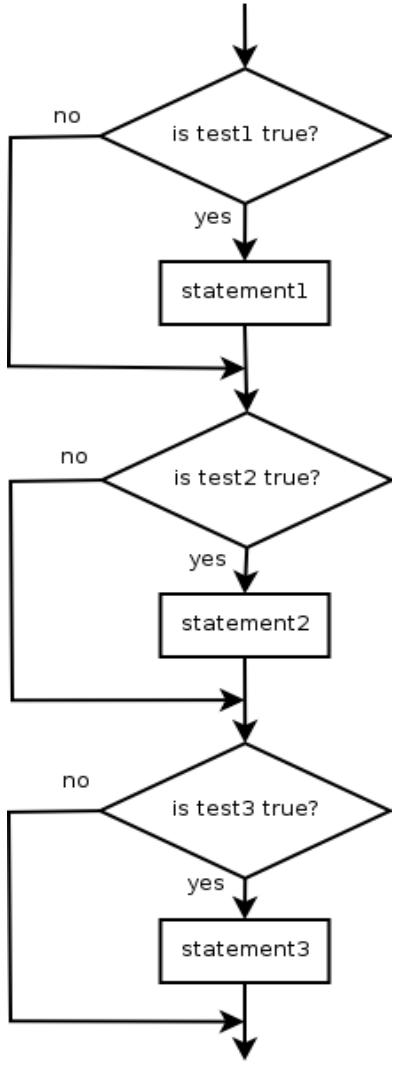
Misuse of `if`

- What's wrong with the following code?

```
Scanner console = new Scanner(System.in);
System.out.print("What percentage did you earn? ");
int percent = console.nextInt();

if (percent >= 90) {
    System.out.println("You got an A!");
}
if (percent >= 80) {
    System.out.println("You got a B!");
}
if (percent >= 70) {
    System.out.println("You got a C!");
}
if (percent >= 60) {
    System.out.println("You got a D!");
}
if (percent < 60) {
    System.out.println("You got an F!");
}
...
```

What will be the output if percent=90?

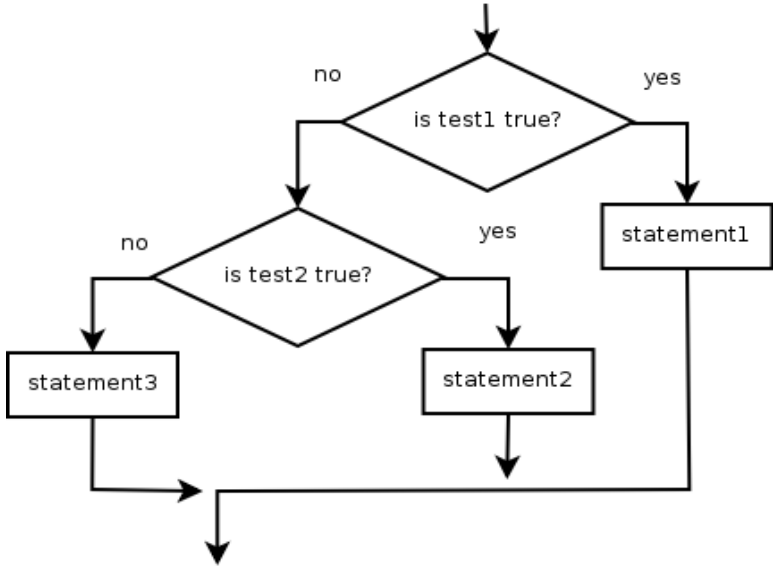


Nested `if/else`

Mutually Exclusive

Chooses between outcomes using many tests

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}
```



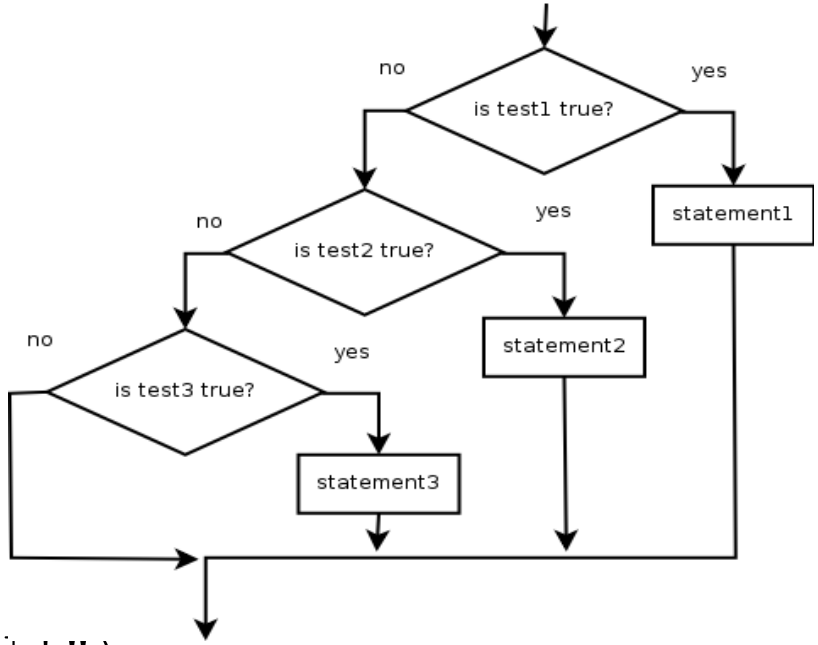
- Example:

```
if (x > 0) {
    System.out.println("Positive");
} else if (x < 0) {
    System.out.println("Negative");
} else {
    System.out.println("Zero");
}
```

Nested `if/else/if` *Mutually Exclusive*

- If it ends with `else`, exactly one path must be taken.
- If it ends with `if`, the code might not execute any path.

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else if (test) {
    statement(s);
}
```



- Example:

```
if (place == 1) {
    System.out.println("Gold medal!");
} else if (place == 2) {
    System.out.println("Silver medal!");
} else if (place == 3) {
    System.out.println("Bronze medal.");
}
```

Nested `if` structures

- exactly 1 path *(mutually exclusive)*

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}
```

- 0 or 1 path *(mutually exclusive)*

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else if (test) {
    statement(s);
}
```

- 0, 1, or many paths *(independent tests; not exclusive)*

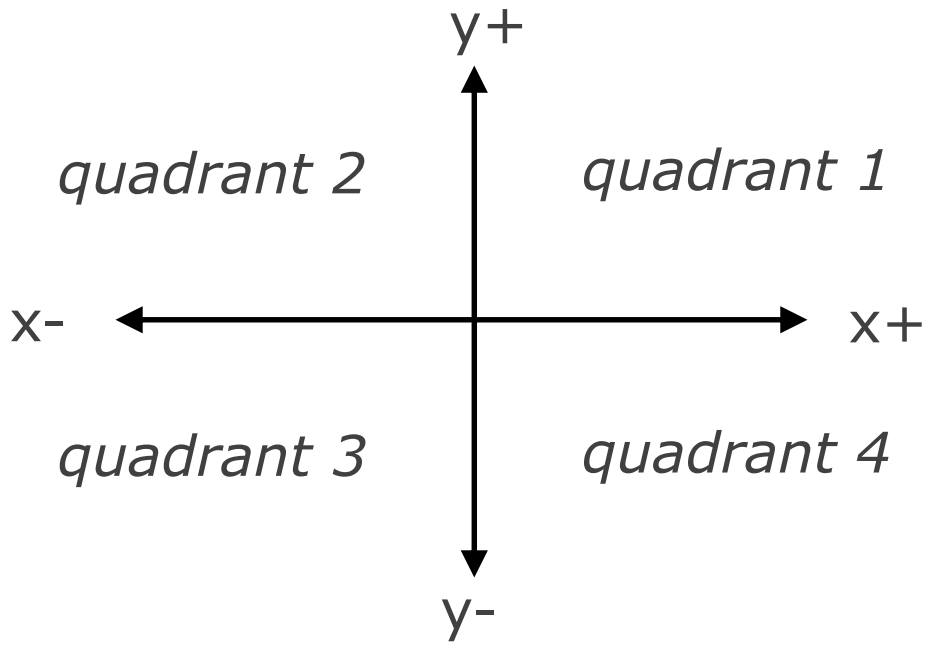
```
if (test) {
    statement(s);
}
if (test) {
    statement(s);
}
if (test) {
    statement(s);
}
```

Which nested `if/else`?

- **(1) if/if/if (2) nested if/else (3) nested if/else/if**
  - Whether a user is lower, middle, or upper-class based on income.
    - **(2)** nested `if / else if / else`
  - Whether you made the dean's list (GPA ≥ 3.8) or honor roll (3.5-3.8).
    - **(3)** nested `if / else if`
  - Whether a number is divisible by 2, 3, and/or 5.
    - **(1)** sequential `if / if / if`
  - Computing a grade of A, B, C, D, or F based on a percentage.
    - **(2)** nested `if / else if / else if / else if / else`

Example1

- Write a program `quadrant` that accepts a pair of real numbers *x* and *y* and prints the quadrant for that point:



  - Example: `(-4.2, 17.3)` is in quadrant `2`
    - If the point falls directly on either axis, return `0`.

Example1 - answer

```
import java.util.*;  // so that I can use Scanner

public class Quadrant {
    public static void main(String[] args) {

    Scanner console = new Scanner(System.in);
    System.out.print("Enter the coordinates of a point:");
    double x = console.nextDouble();
    double y = console.nextDouble();

    if (x > 0 && y > 0) {
        System.out.println("The point is in quadrant 1");
    } else if (x < 0 && y > 0) {
        System.out.println("The point is in quadrant 2");
    } else if (x < 0 && y < 0) {
        System.out.println("The point is in quadrant 3");
    } else if (x > 0 && y < 0) {
        System.out.println("The point is in quadrant 4");
    } else {      // at least one coordinate equals 0
        System.out.println("The point is on axis");
    }
  }
}
```

Example 2

Every Monday thru Friday you go to class. When it is raining you take an umbrella. But on the weekend, what you do depends on the weather. If it is raining you read in bed. Otherwise, you have fun outdoors. Write a program that tells you how to spend your day.

The program that produces output like the following:

```
Enter the day (Use 1 for Monday): 4
Is it raining (Y/N):Y


Go to class
Take an umbrella
```

Example 2 - answer

```
import java.util.*;  // so that I can use Scanner

public class MyDay {
    public static void main(String[] args) {


    Scanner console = new Scanner(System.in);

    System.out.print("Enter the day (use 1 for Monday):");
    int day = console.nextInt();

    System.out.print("Is it raining (Y/N):  ");
    char raining = console.next().charAt(0);


    if  ( ( day == 6) || (day == 7) )  /* Saturday or Sunday */
    {
        if  (raining == 'Y')
            System.out.println("Read in bed");
        else
             System.out.println("Have fun outdoors");
    }
    else
    {
        System.out.println("Go to class ");
        if  (raining == 'Y')
             System.out.println("Take an umbrella");
    }
  }
}
```

Example 3

Formula for body mass index (BMI):

$$BMI = \frac{weight(kg)}{height(m)^2}$$

BMI	Weight class
below 18.5	underweight
18.5 - 24.9	normal
25.0 - 29.9	overweight
30.0 and up	obese

- Write a program that produces output like the following:

```
This program reads data for a person and
computes his body mass index (BMI)

Enter next person's information:
height (in meters)?  70.0
weight (in kilograms)? 194.25

The Person's BMI = 27.868928571428572
overweight
```

Example 3 - answer

```
// This program computes a person's body mass index (BMI) and
// prints whether that person is underweight, normal, overweight or obese

import java.util.*;  // so that I can use Scanner

public class BMI {
    public static void main(String[] args) {
        System.out.println("This program reads data for a person and");
        System.out.println("computes his body mass index (BMI).");
        System.out.println();

        Scanner console = new Scanner(System.in);

        System.out.println("Enter next person's information:");
        System.out.print("height (in meter)? ");
        double height = console.nextDouble();

        System.out.print("weight (in kilograms)? ");
        double weight = console.nextDouble();
        System.out.println();

        double bmi = weight / (height*height);
        System.out.println("The Person's BMI = " + bmi);

        if (bmi < 18.5) {
            System.out.println("underweight");
        } else if (bmi < 25) {
            System.out.println("normal");
        } else if (bmi < 30) {
            System.out.println("overweight");
        } else {
            System.out.println("obese");
        }
  }
```

Comparing strings

- Relational operators such as < and == fail on objects.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name == "Barney") {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

  – This code will compile, but it **will not** print the song.

  – == compares objects by *references* (seen later), so it often gives `false` even when two `String`s have the same letters.

The `equals` method

- Objects are compared using a method named `equals`.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

  - Technically this is a method that returns a value of type `boolean`, the type used in logical tests.
  - Remark:
    - It is preferred to use `if(name.equals("Barney"))` instead of `if(name.equals("Barney")== true)`
    - Similarly, we use `if(!name.equals("Barney"))` instead of `if(name.equals("Barney")== false)`

`String` test methods

Method	Description
`equals(`**str**`)`	whether two strings contain the same characters
`equalsIgnoreCase(`**str**`)`	whether two strings contain the same characters, ignoring upper vs. lower case
`startsWith(`**str**`)`	whether one contains other's characters at start
`endsWith(`**str**`)`	whether one contains other's characters at end
`contains(`**str**`)`	whether the given string is found within this one

```
String name = console.next();
if (name.startsWith("Prof")) {
    System.out.println("When are your office hours?");
} else if (name.equalsIgnoreCase("STUART")) {
    System.out.println("Let's talk about meta!");
}
```

Ternary Operator

- Java ***ternary*** operator lets you assign a value to a variable based on a boolean expression

- General syntax:

```
result = testCondition ? value1 : value2;
```

equivalent to:

```
if (testCondition)
      result = value1;
   else
      result = value2;
```

- Example

```
int max = (a > b) ? a : b;
```