Keenan Knaur
Adjunct Lecturer

California State University, Los Angeles
Computer Science Department

Selection Statements I

CS2011: Introduction to Programming I

Introduction

- If you had a variable called **`radius`** how would you check to make sure that any values entered where not negative?

- ***selection statements*:** statements that let you choose actions with two or more alternative choices and use conditions that are boolean expressions.

- ***Boolean expressions:*** an expression that evaluates to a Boolean value

- ***Boolean value:*** a value that is either **true** or **false**

`boolean` Data Type

`boolean` Data Type

- The **`boolean`** data type is used to declare a variable (Boolean variable) with value of either **`true`** or **`false`**.
  - **`true`** and **`false`** are literals and also reserved words (keywords)

- Syntax:

```
boolean variableName;
```

- Example:

```
boolean isValid = true;
```

Relational Operators

- Used to compare two values, the result of the comparison is Boolean value: **`true`** or **`false`**

- Java has six relation operators:
  - <, <=, >, >=, ==, !=
  - NOTE: the equality operator is a double equal sign

- See Examples:
  - **`BooleanVariablesAndComparisonOperators.java`**
  - **`AdditionQuiz.java`**

`if` Statements

`if` Statements

- if statements execute an action if and only if the ***condition (boolean expression)*** is true.

- Syntax:

```
if (boolean-expression) {
   statement(s);
}
```

- if the **`boolean-expression`** evaluates to **`true`** the statements in the block are executed

- Example:

```
if (radius >= 0) {
   area = radius * radius * PI;
   System.out.println("The area for the circle of  radius " +
     radius + " is " + area);
}
```

`if` Statements

- The boolean-expression of an if statement must be enclosed in parenthesis.
- Caution, block braces can be omitted if the if statement only has one statement inside.
  - **It is always better to have the block braces no matter how many lines of code are inside the if.**
- Forgetting the braces when grouping multiple statements is a common error.
- If you modify the code by adding new statements in an if statement without braces, you will have to insert braces.

```
if (radius >= 0)
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
```

(a) Wrong

```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

(b) Correct

`if-else` Statements

`if-else` Statements

- An **`if-else`** statement decides which statements to execute based on whether the condition is **`true`** or **`false`**.

- A one-way **`if`** statement takes an action if the condition is **`true`**, but does nothing if it is **`false`**.

- The two-way **`if-else`** statement can take an alternative action if the condition evaluates to **`false`** instead of the action when it evaluates to **`true`**.

if-else Statements

- Syntax:

```
if (boolean-expression) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```

- Example:

```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area for the circle   of radius
  " + radius + " is " + area);
}
else {
  System.out.println("Negative input");
}
```

Two-Way `if-else` Statements

- The braces can be omitted if there is only one statement within them, but again, it is always better to have them.

- Example:

```
if (number % 2 == 0)
  System.out.println(number + " is even.");
else
  System.out.println(number + " is odd.");
```

- Code Example: **`IfElseDemo.java`**

Nested `if` and `if-elseif` Statements

Nested `if` and `if-elseif` Statements

- An **`if`** statement can be nested inside another **`if`** statement.
- A statement in an **`if`** or **`if-else`** statement can be any legal Java statement including another **`if`** or **`if-else`** statement.
- There is no limit to the depth of nesting.
- Nesting can be used to implement multiple alternatives.
- Example:

```
if (i > k) {
  if (j > k) {
    System.out.println("i and j are greater than k");
  }
}
else {
  System.out.println("i is less than or equal to k");
}
```

Nested `if` and `if-elseif` Statements

- Instead of using nested **`if-else`** statements, a preferred way would be to use Multi-Way **`if-else`** statements.
- Can have multiple **`else-if`** statements before the **`else`** statement.
- Syntax:

```
if (boolean-expression) {
  statement(s);
}
else if (boolean-expression) {
  statement(s);
}
...
...
else if (boolean-expression) {
  statement(s);
}
else {
  statement(s);
}
```

Nested `if` and `if-elseif` Statements

- These two examples are equivalent but the option on the right is the preferred way to write it using **multi-way if-else statements**.
- Note: Conditions are always tested in order from top to bottom until a condition becomes **`true`** or all of them are **`false`**. A condition is ONLY tested when all of the conditions that come before it are false.

```
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

(a)

Equivalent

This is better

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

(b)

Example

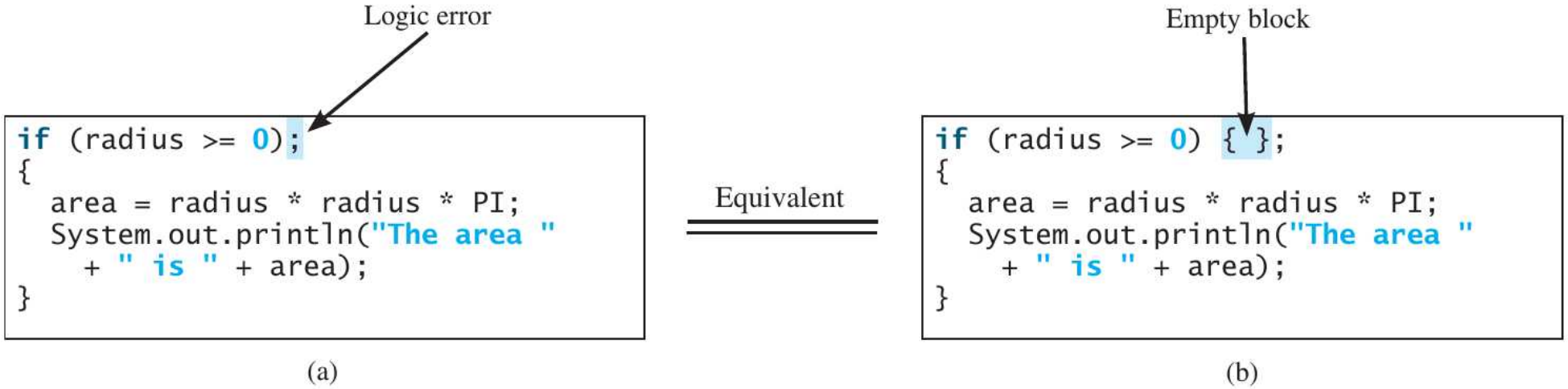
- What is wrong with the following code?

```
if (score >= 60.0)
   grade = 'D';
else if (score >= 70.0)
   grade = 'C';
else if (score >= 80.0)
   grade = 'B';
else if (score >= 90.0)
   grade = 'A';
else
   grade = 'F';
```

Common Errors

Missplaced Semicolon

- Placing a semicolon at the end of an if line is a common mistake.
  - This includes all forms of the if structure.
- Hard to detect (logic error).
- Common with next-line brace style.



Repetitive Testing of Boolean Values

- When you test whether a **boolean** variable is **true** or **false** in a test condition, it is redundant to use the equality comparison operator like in example (a):

```
if (even == true)
  System.out.println(
    "It is even.");
```

(a)

Equivalent

This is better

```
if (even)
  System.out.println(
    "It is even.");
```

(b)

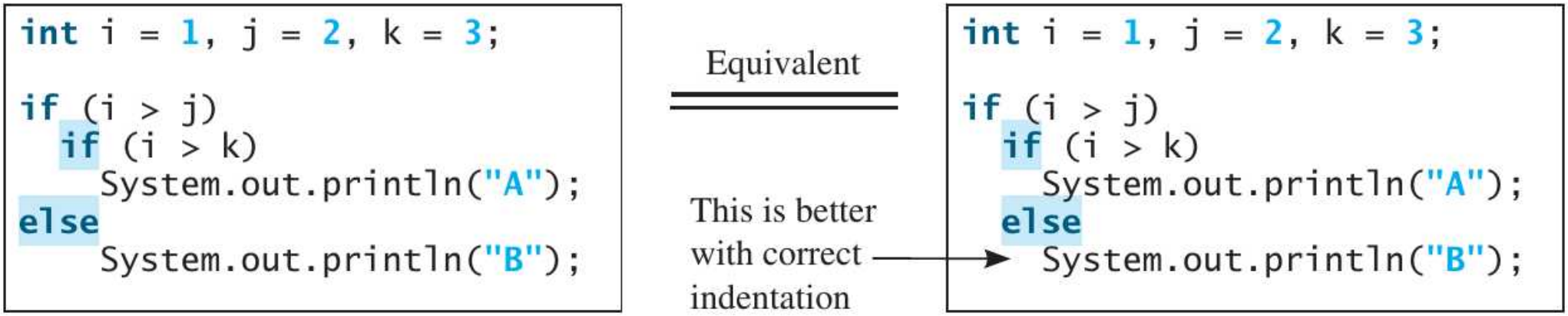
- It is better to test the **boolean** variable directly like in (b). Doing this can also avoid the error where you might use a single (=) instead of the double (==) to compare the equality of two items.

```
if (even = true)
  System.out.println("It is even.");
```

- The statement will not have any compile errors. It will assign **true** to even so that even is always **true**.

Dangling `else` Ambiguity

- The **else** clause will always match the MOST RECENT unmatched **if** clause in the same block



(a) (b)

- The code in (a) has two **if** clauses and one **else** clause and the indentation seems to suggest that the **else** clause matches the first **if** clause.
- In reality, the else clause actually matches the second if clause and this situation is known as ***dangling else ambiguity.***

Dangling `else` Ambiguity

- Nothing will be printed from the previous example because `(i > j)` is false.
- To force the else clause to match the first if clause you have to add a pair of braces to the first if clause:

```
int i = 1, j = 2, k = 3;

if (i > j) {
  if (i > k)
    System.out.println("A");
}
else
  System.out.println("B");
```

Floating-Point Value Equality Testing

- Recall: floating-point numbers have limited precision and can produce round-off errors

- Example: You may think the following should produce **`true`**, but it actually displays **`false`**.

```
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
System.out.println(x == 0.5);
```

  - x is not 0.5 but 0.5000000000000001
  - you cannot reliably test the equality of two floating-point values.

Floating-Point Value Equality Testing

- You can however, test to see if the two values are close enough, by testing whether the difference of the two is less than some threshold value.
- Fact: two numbers **x** and **y** are very close if **|x-y| < ε** for a very small value of ε
  - ε is 10^{-14} for comparing two **double** type values
  - ε is 10^{-7} for comparing two **float** type values

```
final double EPSILON = 1E-14;
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
if (Math.abs(x - 0.5) < EPSILON)
  System.out.println(x + " is approximately 0.5");
```

will display that

```
0.5000000000000001 is approximately 0.5
```

Better Programming: Simplify Boolean Logic

- Often new programmers will write code that assigns a test condition to a boolean variable like the code in example (a).
- The code can be simplified by assigning the result of the test directly to the variable as in example (b).

```
if (number % 2 == 0)
  even = true;
else
  even = false;
```

(a)

Equivalent

This is shorter

```
boolean even
  = number % 2 == 0;
```

(b)

Better Programming: Avoid Duplicate Code in Different Cases

- New programmers can often place duplicate code in different cases which should be combined in one place.

```
if (inState) {
  tuition = 5000;
  System.out.println("The tuition is " + tuition);
}
else {
  tuition = 15000;
  System.out.println("The tuition is " + tuition);
}
```

This is not an error, but it should be better written as follows:

```
if (inState) {
  tuition = 5000;
}
else {
  tuition = 15000;
}
System.out.println("The tuition is " + tuition);
```

Random Numbers

Random Numbers

- **`Math.random()`** returns a random **`double`** value between 0.0 and 1.0 **excluding** 1.0
- To generate a random number between Min and Max the formula is as follows:

```
Min + (int)(Math.random() * (Max - Min + 1))
```

- Example: Generate a random value between 2 and 10:

```
2 + (int)(Math.random() * (10 - 2 + 1))
```

- Code Example:
  - **`RandomNumberGeneration1.java`**
  - **`SubtractionQuiz.java`**

Random Numbers

- Another way is to use the **`Random`** class.
- Import the Random class:

```
import java.util.Random;
```

- Create an instance of the Random class:

```
Random rand = new Random();
```

- Call the **`nextInt(x)`** method through the object you just created.

```
rand.nextInt(x);
```

  - **`nextInt(x)`** returns a random integer value between 0 and x (does not include x).

- To generate a value between min and max use:

```
rand.nextInt((max - min) + 1) + min;
```

- Code Example: **`RandomNumberGeneration2.java`**

References

- Liang, Chapter 03: Selection Statements