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

- 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

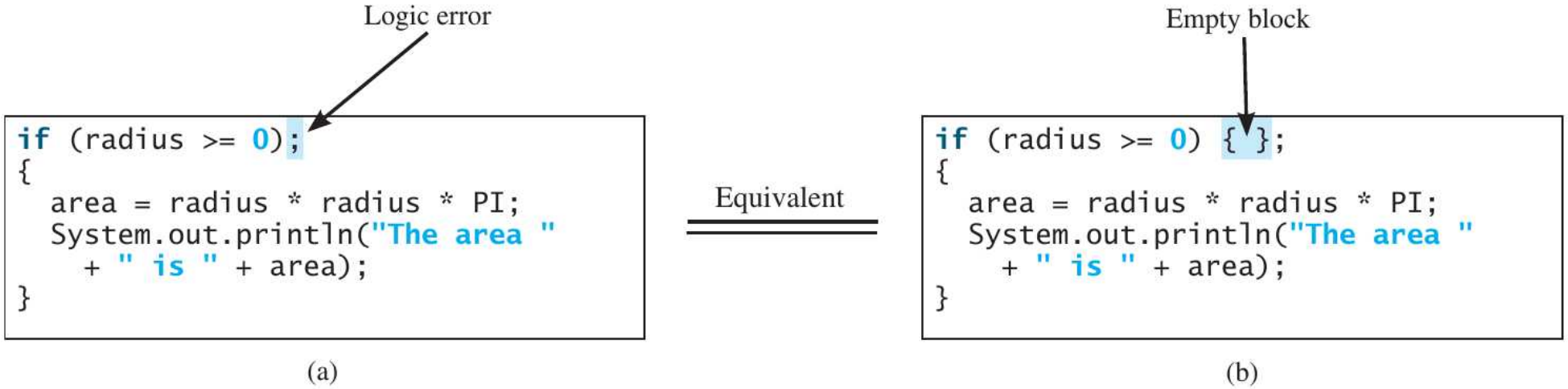
- What is wrong with the following code?

```
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)

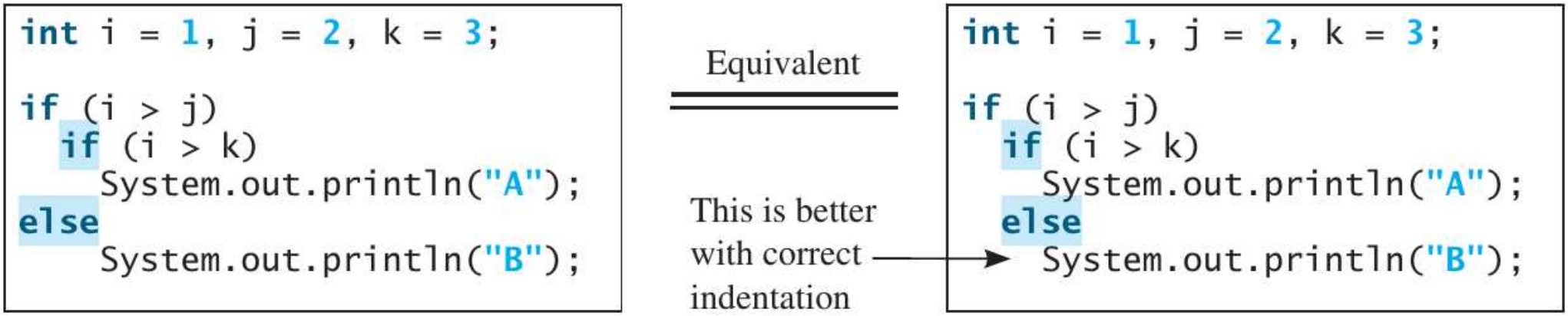
- It is better to test the **boolean** variable directly like in (b). Doing this can also avoid the error where you might use a single (=) instead of the double (==) to compare the equality of two items.

```
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`**

Logical Operators

Logical Operators

- The logical operators `!` (not), `&&` (AND), `||` (OR), and `^` (exclusive OR) can be used to combine multiple conditions to form a compound Boolean expression.

- These ***logical operators***, also known as ***Boolean operators***, operate on Boolean values to create a new Boolean value.

Operator	*Name*	*Description*
!	not	logical negation
&&	and	logical conjunction
||	or	logical disjunction
^	exclusive or	logical exclusion

The ! (NOT) Operator

- Reverses the boolean value, **`true`** becomes **`false`** and **`false`** becomes **`true`**.

e	!e
true	false
false	true

The && (AND Operator)

- Evaluates to **true** if and only if both of the Boolean operands is **true.**
  - If one of the operands is false the entire expression is false.
- ***short-curcuit behavior***
  - When evaluating p1 && p2 Java first evaluates p1 and only will evaluate p2 if p1 is **true**.
  - If p1 is **false** it will not evaluate p2. This helps improve the performance of java.

e_1	e_2	e_1 && e_2
true	true	true
false	false	false
true	false	false
false	true	false

The || (OR) Operator

- Evaluates to true if at least one of the Boolean operands is **true**.
  - If one of the operands is true the entire expression is true.
- ***short-circuit behavior***:
  - When evaluating p1 || p2 Java first evaluates p1 and only will evaluate p2 if p1 is **false**.
  - If p1 is **true** it will not evaluate p2. This helps improve the performance of java.

e_1	e_2	e_1 || e_2
true	true	true
false	false	false
true	false	true
false	true	true

The ^ (Exclusive OR) Operator

- Evaluates to **`true`** if and only if the two operands have OPPOSITE Boolean values.

e_1	e_2	e_1 ^ e_2
true	true	false
false	false	false
true	false	true
false	true	true

`switch` Statements

`switch` Statements

- A **switch** statement executes statements based on the value of a variable or an expression.
- Can be used to replace if and if-else statements when there are many alternatives.
- Can simplify coding for multiple conditions.

`switch` Statements

- Syntax:

```
switch (switch-expression) {
  case value1: statement(s)1;
               break;
  case value2: statement(s)2;
               break;

  ...
  case valueN: statement(s)N;
               break;
  default:     statement(s)-for-default;
}
```

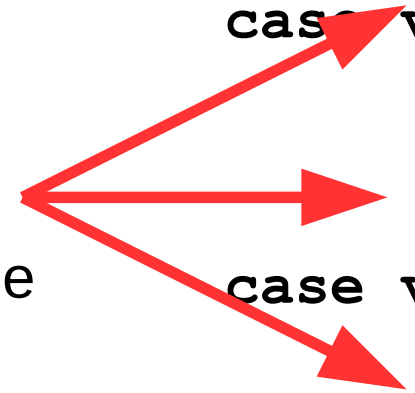
`switch` Statements

- The switch-expression must yield a value of char, byte, short, int, or String type and must always be enclosed in parentheses.

```
switch (switch-expression) {
    case value1:   statement(s)1;
                   break;
    case value2:   statement(s)2;
                   break;
    case valueN:   statement(s)N;
                   break;
    default:       statement(s);
}
```

`switch` Statements

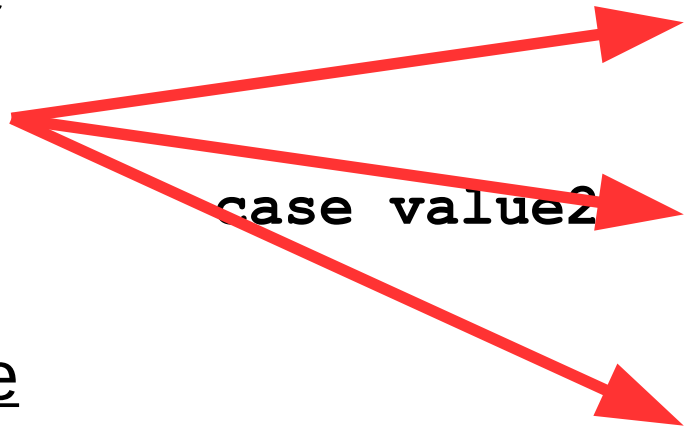
- The value1, ..., and valueN must have the same data type as the value of the switch-expression.
- The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression.
- Note that value1, ..., and valueN are constant expressions, meaning that they cannot contain variables in the expression, such as 1 + x.



```
switch (switch-expression) {

   case value1:   statement(s)1;

                  break;

   case value2:   statement(s)2;

                  break;

   case valueN:   statement(s)N;

                  break;

   default:       statement(s);

}
```

`switch` Statements

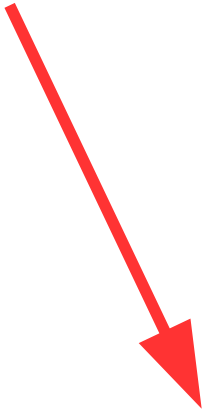
- The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement.
- If the break statement is not present, the next case statement will be executed



```
switch (switch-expression) {

    case value1:   statement(s)1;

                   break;

    case value2:   statement(s)2;

                   break;

    case valueN:   statement(s)N;

                   break;

    default:       statement(s);

}
```

switch Statements

- The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.
- The case statements are executed in sequential order, but the order of the cases (including the default case) does not matter. However, it is good programming style to follow the logical sequence of the cases and place the default case at the end.



```
switch (switch-expression) {

    case value1:   statement(s)1;

                   break;

    case value2:   statement(s)2;

                   break;

    case valueN:   statement(s)N;

                   break;

    default:       statement(s);

}
```

`switch` Statements

- Don't forget to use a **break** statement when one is needed. Once a case is matched the statements starting from the matched case are executed until a **break** statement or the end of the **switch** statement. This is called the *fall-through behavior.*
- Example:

```
switch (ch) {
  case 'a': System.out.println(ch);
  case 'b': System.out.println(ch);
  case 'c': System.out.println(ch);
}
```

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

if-else vs. Statements

```
if (score >= 90)
   grade = 'A';
else if (score >= 80)
   grade = 'B';
else if (score >= 70)
   grade = 'C';
else if (score >= 60)
   grade = 'D';
else
   grade = 'F';
```

```
switch (score / 10) {
  case 10:
  case 9:  grade = 'A';
           break;
  case 8:  grade = 'B';
           break;
  case 7:  grade = 'C';
           break;
  case 6:  grade = 'D';
           break;
```

Conditional Expressions

Conditional Expressions

- A **conditional expression** evaluates an expressions based on a condition.

- A conditional expression can be used to replace a simple **if-else** statement.

- Syntax:

```
boolean-expression ? expression1 : expression2;
```

- The result of the conditional expression is expression1 if boolean-expression is **true**; otherwise the result is expression2

- The first example assigns 1 to y if x is greater than 0, and -1 to y if x is less than or equal to 0.
- The second example is equivalent to the first and does the same thing using a conditional expression

```
if (x > 0)
   y = 1;
else
   y = -1;
```

```
y = (x > 0) ? 1 : -1;
```

- The symbols ? and : together form a conditional operator called a ***ternary operator*** because it uses three operands. It is the ONLY ternary operator in Java.

Conditional Expressions

- Example: Suppose you want to assign the larger number of variable **num1** and **num2** to **max**. You can simply write the following:

```
max = (num1 > num2) ? num1 : num2;
```

- Example: The following statement displays the message "num is even" if **num** is even and otherwise will display "num is odd"

```
System.out.println((num % 2 == 0) ?
  "num is even" : "num is odd");
```

Operator Precedence and Associativity

Operator Precedence and Associativity

- ***Operator precedence*** and ***associativity*** determine the order in which operators are evaluated.
  - parenthesis always evaluated first
  - nested parenthesis evaluated before outer parenthesis
  - operators are evaluated according to the precedence rule and the associativity rule.
  - if two operators with the same precedence are evaluated, the ***associativity*** of the operators determines the order of evaluation.
- All binary operators except assignment operators are ***left-associative*****.**
  - Example: a – b + c – d is equivalent to ((a – b) + c) – d
- The assignment operators are ***right-associative.***
  - Example: a = b += c = 5 is equivalent to a = (b += (c = 5))

Operator Precedence and Associativity

Precedence	*Operator*
↓	`var++` and `var--` (Postfix)
	`+`, `-` (Unary plus and minus), `++var` and `--var` (Prefix)
	(type) (Casting)
	`!` (Not)
	`*`, `/`, `%` (Multiplication, division, and remainder)
	`+`, `-` (Binary addition and subtraction)
	`<`, `<=`, `>`, `>=` (Relational)
	`==`, `!=` (Equality)
	`^` (Exclusive OR)
	`&&` (AND)
	`||` (OR)
	`=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operator)

References

- Liang, Chapter 03: Selection Statements