Conditional Execution

Review: Simple Conditional Execution in Java

```java
if ( <condition> ) { 
    <true block>
} else { 
    <false block>
}
```

- If the condition is true:
  - the statement(s) in the true block are executed
  - the statement(s) in the false block (if any) are skipped

- If the condition is false:
  - the statement(s) in the false block (if any) are executed
  - the statement(s) in the true block are skipped

Computer Science S-111
Harvard University
David G. Sullivan, Ph.D.
Example: Analyzing a Number

```java
Scanner console = new Scanner(System.in);
System.out.print("Enter an integer: ");
int num = console.nextInt();

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

Flowchart for an if-else Statement
**Common Mistake**

- You should **not** put a semi-colon after an if-statement header:

```java
if (num % 2 == 0); {
    System.out.println(...);
    ...
}
```

- The semi-colon ends the `if` statement.
- thus, it has an empty true block
- The `println` and other statements are independent of the `if` statement, and always execute.

**Choosing at Most One of Several Options**

- Consider this code:

```java
if (num < 0) {
    System.out.println("The number is negative.");
}
if (num > 0) {
    System.out.println("The number is positive.");
}
if (num == 0) {
    System.out.println("The number is zero.");
}
```

- All three conditions are evaluated, but at most one of them can be true (in this case, *exactly* one).
Choosing at Most One of Several Options (cont.)

- We can do this instead:
  
  ```java
  if (num < 0) {
    System.out.println("The number is negative.");
  }
  else if (num > 0) {
    System.out.println("The number is positive.");
  }
  else if (num == 0) {
    System.out.println("The number is zero.");
  }
  ```

- If the first condition is true, it will skip the second and third.
- If the first condition is false, it will evaluate the second, and if the second condition is true, it will skip the third.
- If the second condition is false, it will evaluate the third, etc.

Choosing at Most One of Several Options (cont.)

- We can also make things more compact as follows:
  
  ```java
  if (num < 0) {
    System.out.println("The number is negative.");
  } else if (num > 0) {
    System.out.println("The number is positive.");
  } else if (num == 0) {
    System.out.println("The number is zero.");
  }
  ```

- This emphasizes that the entire thing is one compound statement.
if-else if Statements

• Syntax:

```java
if (<condition1>) {
    <true block for condition1>
} else if (<condition2>) {
    <true block for condition2>
}
...
} else {
    <false block for all of the conditions>
}
```

• The conditions are evaluated in order. The true block of the first true condition is executed. All of the remaining conditions and their blocks are skipped.

• If no condition is true, the false block (if any) is executed.

Flowchart for an if-else if Statement

```
condition1
  true
  |     
  v     v
true block 1
false

condition2
  true
  |     
  v     v
true block 2
false

... false

false block

false

next statement
```
Choosing Exactly One Option

• Consider again this code fragment:

```java
if (num < 0) {
    System.out.println("The number is negative.");
} else if (num > 0) {
    System.out.println("The number is positive.");
} else if (num == 0) {
    System.out.println("The number is zero.");
}
```

• One of the conditions must be true, so we can omit the last one:

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

Types of Conditional Execution

• If it want to execute any number of several conditional blocks, use sequential if statements:

```java
if (num < 0) {
    System.out.println("The number is negative.");
}
if (num % 2 == 0) {
    System.out.println("The number is even.");
}
```

• If you want to execute at most one (i.e., 0 or 1) of several blocks, use an if-elseif statement ending in elseif:

```java
if (num < 0) {
    System.out.println("The number is negative.");
} else if (num > 0) {
    System.out.println("The number is positive.");
}
```

• If you want to execute exactly one of several blocks, use an if-elseif ending in just else (see bottom of last slide).
Find the Logic Error

Scanner console = new Scanner(System.in);
System.out.print("Enter the student's score: ");
int score = console.nextInt();

String grade;
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";
}

Review: Variable Scope

- Recall: the scope of a variable is the portion of a program in which the variable can be used.

- By default, the scope of a variable:
  - begins at the point at which it is declared
  - ends at the end of the innermost block that encloses the declaration

- Because of these rules, a variable cannot be used outside of the block in which it is declared.
Variable Scope and if-else statements

- The following program will produce compile-time errors:

```java
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    System.out.print("enter a positive int: ");
    int num = console.nextInt();
    if (num < 0) {
        System.out.println("number is negative; " + "using its absolute value");
        double sqrt = Math.sqrt(num * -1);
    } else {
        sqrt = Math.sqrt(num);
    }
    System.out.println("square root = " + sqrt);
}
```

- Why?

Variable Scope and if-else statements (cont.)

- To eliminate the errors, declare the variable outside of the true block:

```java
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    System.out.print("enter a positive int: ");
    int num = console.nextInt();
    double sqrt;
    if (num < 0) {
        System.out.println("number is negative; " + "using its absolute value");
        sqrt = Math.sqrt(num * -1);
    } else {
        sqrt = Math.sqrt(num);
    }
    System.out.println("square root = " + sqrt);
}
```

- What is the scope of sqrt now?
Review: Loop Patterns for n Repetitions

- Thus far, we’ve mainly used for loops to repeat something a definite number of times.

- We’ve seen two different patterns for this:
  - pattern 1:
    ```java
    for (int i = 0; i < n; i++) {
        <statements to repeat>
    }
    ```
  - pattern 2:
    ```java
    for (int i = 1; i <= n; i++) {
        <statements to repeat>
    }
    ```

Another Loop Pattern: Cumulative Sum

- We can also use a for loop to add up a set of numbers.

- Basic pattern (using pseudocode):
  ```
  sum = 0
  for (all of the numbers that we want to sum) {
      num = the next number
      sum = sum + num
  }
  ```
Example of Using a Cumulative Sum

```java
public class GradeAverager {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("number of grades? ");
        int numGrades = console.nextInt();
        if (numGrades <= 0) {
            System.out.println("nothing to average");
        } else {
            int sum = 0;
            for (int i = 1; i <= numGrades; i++) {
                System.out.print("grade "+i+": ");
                int grade = console.nextInt();
                sum = sum + grade;
            }
            System.out.println("The average is "+
                    (double)sum / numGrades);
        }
    }
}
```

• Note the use of an if-else statement to handle invalid user inputs.

Tracing Through a Cumulative Sum

• Let's trace through this code.
  
  ```java
  int sum = 0;
  for (int i = 1; i <= numGrades; i++) {
      System.out.print("grade "+i+": ");
      int grade = console.nextInt();
      sum = sum + grade;
  }
  
  assuming that the user enters these grades: 80, 90, 84.
  
  numGrades = 3
  
  i i <= numGrades grade sum
  ```
Conditional Execution and Return Values

• With conditional execution, it's possible to write a method with more than one `return` statement.

  • example:
    
    ```java
    public static int min(int a, int b) {
      if (a < b) {
        return a;
      } else {
        return b;
      }
    }
    ```
    
  • Only one of the `return` statements is executed.

  • As soon as you reach a return statement, the method's execution stops and the specified value is returned.
    • the rest of the method is not executed

Conditional Execution and Return Values (cont.)

• Instead of writing the method this way:
  
  ```java
  public static int min(int a, int b) {
    if (a < b) {
      return a;
    } else {
      return b;
    }
  }
  ```

  we could instead write it like this, without the `else`:
  
  ```java
  public static int min(int a, int b) {
    if (a < b) {
      return a;
    } else {
      return b;
    }
  }
  ```

  • Why is this equivalent?
Conditional Execution and Return Values (cont.)

- Consider this method, which has a compile-time error:

```java
public static int compare(int a, int b) {
    if (a < b) {
        return -1;
    } else if (a > b) {
        return 1;
    } else if (a == b) {
        return 0;
    }
}
```

- Because all of the return statements are connected to conditions, the compiler worries that no value will be returned.

Conditional Execution and Return Values (cont.)

- Here’s one way to fix it:

```java
public static int compare(int a, int b) {
    if (a < b) {
        return -1;
    } else if (a > b) {
        return 1;
    } else {
        return 0;
    }
}
```
Conditional Execution and Return Values (cont.)

• Here’s another way:
  ```java
  public static int compare(int a, int b) {
    if (a < b) {
      return -1;
    } else if (a > b) {
      return 1;
    }
    return 0;
  }
  ```

• Both fixes allow the compiler to know for certain that a value will always be returned.

Returning From a `void` Method

```java
  public static void repeat(String msg, int n) {
    if (n <= 0) { // special cases
      return;
    }
    for (int i = 0; i < n; i++) {
      System.out.println(msg);
    }
  }
```

• Note that this method has a return type of `void`.
  • it doesn’t return a value.

• However, it still has a `return` statement.
  • used to break out of the method
  • note that there’s nothing between the `return` and the ;
Testing for Equivalent Primitive Values

- The `==` and `!=` operators are used when comparing primitives.
  - `int`, `double`, `char`, etc.

- Example:
  ```java
  Scanner console = new Scanner(System.in);
  ...
  System.out.print("Do you have another (y/n)? ");
  char choice = console.next().charAt(0);
  if (choice == 'y') { // this works just fine
    processItem();
  } else if (choice == 'n') {
    return;
  } else {
    System.out.println("invalid input");
  }
  ```

Testing for Equivalent Objects

- The `==` and `!=` operators do not typically work when comparing objects. (We'll see why this is later.)

- Example:
  ```java
  Scanner console = new Scanner(System.in);
  System.out.print("regular or diet? ");
  String choice = console.next();
  if (choice == "regular") { // doesn't work
    processRegular();
  } else {
    ...
  }
  ```

- `choice == "regular"` compiles, but it evaluates to `false`, even when the user does enter "regular"!
Testing for Equivalent Objects (cont.)

• We use a special method called the `equals` method to test if two objects are equivalent.
  
  • `choice.equals("regular")` compares the string represented by the variable `choice` with the string "regular"
  • returns `true` when they are equivalent
  • returns `false` when they are not

equalsIgnoreCase()

• We often want to compare two strings without paying attention to the case of the letters.
  
  • example: we want to treat as equivalent:
    "regular"
    "Regular"
    "REGULAR"
    etc.

  • The `String` class has a method called `equalsIgnoreCase` that can be used for this purpose:
    ```java
    if (choice.equalsIgnoreCase("regular")) {
      ...
    }
    ```
Example Problem: Ticket Sales

- Different prices for balcony seats and orchestra seats

- Here are the rules:
  - persons younger than 25 receive discounted prices:
    - $20 for balcony seats
    - $35 for orchestra seats
  - everyone else pays the regular prices:
    - $30 for balcony seats
    - $50 for orchestra seats

- Assume only valid inputs.

Ticket Sales Program: `main` method

```java
Scanner console = new Scanner(System.in);
System.out.print("Enter your age: ");
int age = console.nextInt();
if (age < 25) {
    // handle people younger than 25
    System.out.print("orchestra or balcony? ");
    String choice = console.next();
    int price;
    if (choice.equalsIgnoreCase("orchestra")) {
        price = 35;
    } else {
        price = 20;
    }
    System.out.println("The price is $" + price);
} else {
    // handle people 25 and older
    ...
}
```
Ticket Sales Program: main method (cont.)

...  
} else {
    // handle people 25 and older  
    System.out.print("orchestra or balcony? ");  
    String choice = console.next();  
    int price;  
    if (choice.equalsIgnoreCase("orchestra")) {
        price = 50;  
    } else {
        price = 30;  
    }
    System.out.println("The price is $\) + price);
}

Where Is the Code Duplication?

...  
if (age < 25) {  
    System.out.print("orchestra or balcony? ");  
    String choice = console.next();  
    int price;  
    if (choice.equalsIgnoreCase("orchestra")) {
        price = 35;  
    } else {  
        price = 20;  
    }
    System.out.println("The price is $\) + price);
} else {
    System.out.print("orchestra or balcony? ");  
    String choice = console.next();  
    int price;  
    if (choice.equalsIgnoreCase("orchestra")) {
        price = 50;  
    } else {
        price = 30;  
    }
    System.out.println("The price is $\) + price);
}
Factoring Out Code Common to Multiple Cases

Scanner console = new Scanner(System.in);
int age = console.nextInt();
System.out.print("Enter your age: ");
int price;
if (choice.equalsIgnoreCase("orchestra")) {
    price = 35;
} else {
    price = 20;
}
else {
    price = 50;
}
else {
    price = 30;
}
System.out.println("The price is "+price);

What Other Change Is Needed?

Scanner console = new Scanner(System.in);
int age = console.nextInt();
System.out.print("Enter your age: ");
int price;
if (choice.equalsIgnoreCase("orchestra")) {
    price = 35;
} else {
    price = 20;
}
else {
    price = 50;
}
else {
    price = 30;
}
System.out.println("The price is "+price);
public static void main(String[] args) {
    int age = console.nextInt();
    System.out.print("orchestra or balcony? ");
    String choice = console.next();
    int price;
    if (age < 25) {
        price = 20;
    } else {
        price = 35;
    }
    System.out.println("The price is $" + price);
}

public static _______ discountPrice(__________________) {
}

Expanded Ticket Sales Problem

• One additional case:
  • persons younger than 13 cannot buy a ticket
  • persons whose age is 13-24 receive discounted prices:
    • $20 for balcony seats
    • $35 for orchestra seats
  • everyone else pays the regular prices:
    • $30 for balcony seats
    • $50 for orchestra seats
Here's the Unfactored Version

```java
if (age < 13) {
    System.out.println("You cannot buy a ticket.'");
} else if (age < 25) {
    System.out.print("orchestra or balcony? ");
    String choice = console.next();
    int price;
    if (choice.equalsIgnoreCase("orchestra")) {
        price = 35;
    } else {
        price = 20;
    }
    System.out.println("The price is "+ price);
} else {
    System.out.print("orchestra or balcony? ");
    String choice = console.next();
    int price;
    if (choice.equalsIgnoreCase("orchestra")) {
        price = 50;
    } else {
        price = 30;
    }
    System.out.println("The price is "+ price);
}
```

We now have code common to the 2nd and 3rd cases, but not the 1st.

---

Group the Second and Third Cases Together

```java
... 
if (age < 13) {
    System.out.println("You cannot buy a ticket.'");
} else {
    if (age < 25) {
        System.out.print("orchestra or balcony? ");
        String choice = console.next();
        int price;
        if (choice.equalsIgnoreCase("orchestra")) {
            price = 35;
        } else {
            price = 20;
        }
        System.out.println("The price is "+ price);
    } else {
        System.out.print("orchestra or balcony? ");
        String choice = console.next();
        ... 
        System.out.println("The price is "+ price);
    }
}
```
Then Factor Out the Common Code

```java
... 
if (age < 13) {
    System.out.println("You cannot buy a ticket.");
} else {
    System.out.print("orchestra or balcony? ");
    String choice = console.next();
    int price;
    if (age < 25) {
        if (choice.equalsIgnoreCase("orchestra")) {
            price = 35;
        } else {
            price = 20;
        }
    } else {
        if (choice.equalsIgnoreCase("orchestra")) {
            price = 50;
        } else {
            price = 30;
        }
    }
    System.out.println("The price is $" + price);
}
```

Case Study: Coffee Shop Price Calculator

- Relevant info:
  - brewed coffee prices by size:
    - tiny: $1.60
    - medio: $1.80
    - gigundo: $2.00
  - latte prices by size:
    - tiny: $2.80
    - medio: $3.20
    - gigundo: $3.60
  - **plus**, add 50 cents for a latte with flavored syrup
- sales tax:
  - students: no tax
  - non-students: 6.25% tax
Case Study: Coffee Shop Price Calculator (cont.)

- Developing a solution:
  1. Begin with an *unstructured* solution.
     - everything in the `main` method
     - use if-else-if statement(s) to handle the various cases
  2. Next, *factor out* code that is common to multiple cases.
     - put it either before or after the appropriate if-else-if statement
  3. Finally, create a fully *structured* solution.
     - use procedural decomposition to capture logical pieces of the solution
Optional: Comparing Floating-Point Values

- Because the floating-point types have limited precision, it's possible to end up with *roundoff errors*.

- Example:
  
  ```java
  double sum = 0.1 + 0.1 + 0.1 + 0.1 + 0.1;
  sum = sum + 0.1 + 0.1 + 0.1 + 0.1 + 0.1;
  System.out.println(sum);
  // get 0.9999999999999999!
  ```

- Thus when trying to determine if two floating-point values are equal, we usually do *not* use the `==` operator.

- Instead, we test if the difference between the two values is less than some small *threshold* value:

  ```java
  if (Math.abs(sum - 1.0) < 0.0000001) {
    System.out.println(sum + " == 1.0");
  }
  ```

Optional: Another Cumulative Computation

- The same pattern can be used for other types of computations.

- Example: counting the occurrences of a character in a string.

- Let's write a static method called `numOccur` that does this.

  ```java
  public static ___ numOccur(_____________________) {
    numOccur('l', "hello") should return 2
    numOccur('s', "Mississippi") should return 4
  }
  ```