Conditionals and Loops

• Now we will examine programming statements that allow us to:
  – make decisions
  – repeat processing steps in a loop

• Chapter 5 focuses on:
  – boolean expressions
  – the if and if-else statements
  – comparing data
  – while loops
  – iterators
  – more drawing techniques
  – more GUI components
Outline

- Boolean Expressions
  - The if Statement
  - Comparing Data
  - The while Statement
  - Iterators
  - The ArrayList Class
- Determining Event Sources
- Check Boxes and Radio Buttons
Flow of Control

- Unless specified otherwise, the order of statement execution through a method is linear: one after another

- Some programming statements allow us to make decisions and perform repetitions

- These decisions are based on boolean expressions (also called conditions) that evaluate to true or false

- The order of statement execution is called the flow of control
Conditional Statements

• A *conditional statement* lets us choose which statement will be executed next

• They are sometimes called *selection statements*

• Conditional statements give us the power to make basic decisions

• The Java conditional statements are the:
  – *if* and *if-else* statement
  – *switch* statement

• We'll explore the switch statement in Chapter 6
Boolean Expressions

• A condition often uses one of Java's *equality operators* or *relational operators*, which all return boolean results:

  
  `==`   equal to
  `!=`   not equal to
  `<`    less than
  `>`    greater than
  `<=`   less than or equal to
  `>=`   greater than or equal to

• Note the difference between the equality operator (`==`) and the assignment operator (`=`)
Boolean Expressions

• An if statement with its boolean condition:

```java
if (sum > MAX)
    delta = sum - MAX;
```

• First, the condition is evaluated: the value of `sum` is either greater than the value of `MAX`, or it is not

• If the condition is true, the assignment statement is executed; if it isn't, it is skipped

• See `Age.java`
import java.util.Scanner;

public class Age {
    public static void main (String[] args) {
        final int MINOR = 21;

        Scanner scan = new Scanner (System.in);

        System.out.print ("Enter your age: ");
        int age = scan.nextInt();
        continue
System.out.println("You entered: " + age);

if (age < MINOR)
   System.out.println("Youth is a wonderful thing. Enjoy.");

System.out.println("Age is a state of mind.");
}
```java
System.out.println("You entered: "+ age);

if (age < MINOR)
    System.out.println("Youth is a wonderful thing. Enjoy.");

System.out.println("Age is a state of mind.");
```

**Sample Run**

Enter your age: 47  
You entered: 47  
Age is a state of mind.

**Another Sample Run**

Enter your age: 12  
You entered: 12  
Youth is a wonderful thing. Enjoy.  
Age is a state of mind.
Quiz

• What is the value of x, a, and b after the execution of the following statements?

```c
int x=1, a=2, b=3;
if (x < 0)
b=5;
a= b*2;
x=4;
```
Quiz

• What is the value of x, a, and b after the execution of the following statements?

```c
int x=1, a=2, b=3;
if (x < 0)
  b=5;  // this statement is not executed
a= b*2;
x=4;
```

x is 4
a is 6
b is 3
Logical Operators

• Boolean expressions can also use the following logical operators:
  
  ! Logical NOT
  && Logical AND
  || Logical OR

• They all take boolean operands and produce boolean results

• Logical NOT is a unary operator (it operates on one operand)

• Logical AND and logical OR are binary operators (each operates on two operands)
Logical NOT

• The *logical NOT* operation is also called *logical negation* or *logical complement*.

• If some boolean condition \( a \) is true, then \( !a \) is false; if \( a \) is false, then \( !a \) is true.

• Logical expressions can be shown using a *truth table*:

<table>
<thead>
<tr>
<th></th>
<th>!a</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
Quiz

• What is the value of x, a, and b after the execution of the following statements?

```c
int x=0, a=2, b=3;
if (!(x >1))
    b=5;
    a= b*2;
    x=4;
```
Quiz

• What is the value of x, a, and b after the execution of the following statements?

```java
int x=0, a=2, b=3;
if (!(x >1))
b=5; // this statement is executed
a= b*2;
x=4;
```

x is 4
a is 10
b is 5
Logical AND and Logical OR

• The *logical AND* expression
  
  \[ a \land b \]

  is true if both *a* and *b* are true, and false otherwise

• The *logical OR* expression

  \[ a \lor b \]

  is true if *a* or *b* or both are true, and false otherwise
Logical AND and Logical OR

- A truth table shows all possible true-false combinations of the terms.

- Since `&&` and `||` each have two operands, there are four possible combinations of conditions `a` and `b`.

| a    | b    | a && b | a || b |
|------|------|--------|--------|
| true | true | true   | true   |
| true | false| false  | true   |
| false| true | false  | true   |
| false| false| false  | false  |
Logical Operators

• Expressions that use logical operators can form complex conditions

```java
if (total < MAX+5 && !found)
    System.out.println ("Processing...");
```

• All **logical operators** have **lower precedence** than the **relational operators**

• The `!` operator has higher precedence than `&&` and `||`
Quiz

• Are the two following Boolean expressions equivalent? Assume that \( a \) and \( b \) are Boolean variables.

1) \(! (a \ &\ & \ b)\)
2) \(! a \ &\ & \ b\)
Quiz

• Are the two following Boolean expressions equivalent? Assume that $a$ and $b$ are Boolean variables.

1) $! (a \&\& b)$
2) $!a \&\& b$

NO, the first is always true except when $a$ and $b$ are true. The second is true only when $a$ is false and $b$ is true.
Boolean Expressions

- Specific expressions can be evaluated using truth tables

<table>
<thead>
<tr>
<th>total &lt; MAX</th>
<th>found</th>
<th>!found</th>
<th>total &lt; MAX &amp;&amp; !found</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Quiz

• Assuming that \( c_1 \) and \( c_2 \) are boolean variables, create a truth table for the expression:

\[
(c_1 \land \neg c_2) \lor (\neg c_1 \land c_2)
\]
Quiz

- Assuming that \(c_1\) and \(c_2\) are boolean variables, create a truth table for the expression:

\[
(c_1 \land \neg c_2) \lor (\neg c_1 \land c_2)
\]

<table>
<thead>
<tr>
<th>(c_1)</th>
<th>(c_2)</th>
<th>(\neg c_1)</th>
<th>(\neg c_2)</th>
<th>(c_1 \land \neg c_2)</th>
<th>(\neg c_1 \land c_2)</th>
<th>(c_1 \land \neg c_2 \lor \neg c_1 \land c_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Short-Circuited Operators

• The processing of && and || is “short-circuited”

• If the left operand is sufficient to determine the result, the right operand is not evaluated

    if (count != 0 && total/count > MAX)
    System.out.println ("Testing.");

• This type of processing should be used carefully
Quiz

• What is the value of count and base after the execution of the following statements?

```cpp
int count=2, base=0;
if (count > 1 || ++base > count) 
    count--; 
```
Quiz

• What is the value of count and base after the execution of the following statements?

```java
int count=2, base=0;
if (count > 1 || ++base > count)
    count--;

count is 1
base is 0
```
Quiz

• What is the value of count and base after the execution of the following statements?

```java
int count=2, base=0;
if (++base > count || count > 1)
    count--;
```
Quiz

- What is the value of count and base after the execution of the following statements

```java
int count=2, base=0;
if (++base > count || count > 1)
    count--;

count is 1
base is 1
```
Outline

Boolean Expressions
The if Statement
Comparing Data
The while Statement
Iterators
The ArrayList Class
Determining Event Sources
Check Boxes and Radio Buttons
The if Statement

- Let's now look at the `if` statement in more detail
- The `if` statement has the following syntax:

```java
if (condition) statement;
```

The `condition` must be a boolean expression. It must evaluate to either true or false.

If the `condition` is true, the `statement` is executed. If it is false, the `statement` is skipped.
Logic of an if statement

Flowchart
Indentation

• The statement controlled by the `if` statement is **indented** to indicate that relationship.

• The use of a consistent indentation style makes a program easier to read and understand.

• The **compiler ignores indentation**, which can lead you to make (logical) errors if the indentation is not correct.

"Always code as if the person who ends up maintaining your code will be a violent psychopath who knows where you live."

-- Martin Golding
Quick Check

What do the following statements do?

```java
if (total != stock + warehouse)
    inventoryError = true;

if (found || !done)
    System.out.println("Ok");
```
Quick Check

What do the following statements do?

```java
if (total != stock + warehouse)
    inventoryError = true;
```

Sets the boolean variable to true if the value of `total` is not equal to the sum of `stock` and `warehouse`.

```java
if (found || !done)
    System.out.println("Ok");
```

Prints "Ok" if `found` is true or `done` is false.
The if-else Statement

• An else clause can be added to an if statement to make an if-else statement

```java
if ( condition )
    statement1;
else
    statement2;
```

• If the condition is true, statement1 is executed; if the condition is false, statement2 is executed

• One or the other will be executed, but not both

• See Wages.java
import java.text.NumberFormat;
import java.util.Scanner;

public class Wages
{
    // Demonstrates the use of an if-else statement.
    public static void main (String[] args)
    {
        final double RATE = 8.25;  // regular pay rate
        final int STANDARD = 40;   // standard hours in a work week

        Scanner scan = new Scanner (System.in);

        double pay = 0.0;

        continue
System.out.print("Enter the number of hours worked: ");
int hours = scan.nextInt();

System.out.println();

// Pay overtime at "time and a half"
if (hours > STANDARD)
    pay = STANDARD * RATE + (hours-STANDARD) * (RATE * 1.5);
else
    pay = hours * RATE;

NumberFormat fmt = NumberFormat.getCurrencyInstance();
System.out.println("Gross earnings: "+fmt.format(pay));
}
System.out.print( "Enter the number of hours worked: ");
int hours = scan.nextInt();

System.out.println();

// Pay overtime at "time and a half"
if (hours > STANDARD)
    pay = STANDARD * RATE + (hours-STANDARD) * (RATE * 1.5);
else
    pay = hours * RATE;

NumberFormat fmt = NumberFormat.getCurrencyInstance();
System.out.println ("Gross earnings: "+ fmt.format(pay));
Logic of an if-else statement

condition evaluated

true

statement1

false

statement2
Quiz

What is printing the program described in this flowchart?

Start

READ N

M = 1
F = 1

F = F * M

M = M + 1

IS M = N?

YES

PRINT F

NO

END
If Statement

```
if (Expression) Statement
else Statement
```
The Coin Class

• Let's look at an example that uses a class that represents a coin that can be flipped

• Instance data is used to indicate which face (heads or tails) is currently showing

• See CoinFlip.java
• See Coin.java
public class CoinFlip
{
    public static void main (String[] args)
    {
        Coin myCoin = new Coin();

        myCoin.flip();

        System.out.println (myCoin);

        if (myCoin.isHeads())
            System.out.println ("You win.");
        else
            System.out.println ("Better luck next time.");
    }
}
public class CoinFlip
{
    public static void main (String[] args)
    {
        Coin myCoin = new Coin();

        myCoin.flip();

        System.out.println (myCoin);

        if (myCoin.isHeads())
            System.out.println ("You win.");
        else
            System.out.println ("Better luck next time.");
    }
}
public class Coin {
    private final int HEADS = 0;
    private final int TAILS = 1;

    private int face;

    // Sets up the coin by flipping it initially.
    public Coin () {
        flip();
    }

    continue
//-------------------------------
//  Flips the coin by randomly choosing a face value.
//-------------------------------
public void flip ()
{
    face = (int) (Math.random() * 2);
}

//-------------------------------
//  Returns true if the current face of the coin is heads.
//-------------------------------
public boolean isHeads ()
{
    return (face == HEADS);
}
public String toString()
{
    String faceName;

    if (face == HEADS)
        faceName = "Heads";
    else
        faceName = "Tails";

    return faceName;
}
Indentation Revisited

• Remember that indentation is for the human reader, and is ignored by the compiler

```java
if (depth >= UPPER_LIMIT)
    delta = 100;
else
    System.out.println("Reseting Delta");
    delta = 0;
```

• Despite what the indentation implies, `delta` will be set to 0 no matter what
Consider the following code that will assign a letter grade of 'A', 'B', 'C', 'D', or 'F' depending on a student's test score.

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

A) This code will work correctly in all cases
B) This code will work correctly only if grade >= 60
C) This code will work correctly only if grade < 60
D) This code will work correctly only if grade < 70
E) This code will not work correctly under any circumstances
Consider the following code that will assign a letter grade of 'A', 'B', 'C', 'D', or 'F' depending on a student's test score.

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

A) This code will work correctly in all cases
B) This code will work correctly only if grade >= 60
C) This code will work correctly only if grade < 60
D) This code will work correctly only if grade < 70
E) This code will not work correctly under any circumstances
Block Statements

• Several statements can be grouped together into a *block statement* delimited by braces

• A *block statement* can be used wherever a *statement is called for* in the Java syntax rules

```java
if (total > MAX)
{
    System.out.println ("Error!!");
    errorCount++;  
}
```
Block Statements

• The *if* clause, or the *else* clause, or both, could govern block statements

```java
if (total > MAX)
{
    System.out.println("Error!!");
    errorCount++;  
}
else
{
    System.out.println("Total: " + total);
    current = total*2;
}
```

• See *Guessing.java*
import java.util.*;

public class Guessing
{
  // Plays a simple guessing game with the user.
  public static void main (String[] args)
  {
    final int MAX = 10;
    int answer, guess;

    Scanner scan = new Scanner (System.in);
    Random generator = new Random();

    answer = generator.nextInt(MAX) + 1;

    continue
continue

    System.out.print ("I'm thinking of a number between 1 and " + MAX + ". Guess what it is: ");

    guess = scan.nextInt();

    if (guess == answer)
        System.out.println("You got it! Good guessing!");
    else
    {
        System.out.println("That is not correct, sorry.");
        System.out.println("The number was " + answer);
    }
}
Sample Run

I'm thinking of a number between 1 and 10. Guess what it is: 6
That is not correct, sorry.
The number was 9

if (guess == answer)
    System.out.println("You got it! Good guessing!");
else
{
    System.out.println("That is not correct, sorry.");
    System.out.println("The number was "+answer);
}

Nested if Statements

• The statement executed as a result of an if or else clause could be another if statement

• These are called *nested if statements*

• Example:

```java
if (age < 21)
    if (age > 2)
        System.out.println("age is larger than " + " 2 and smaller than 21");
```
Nested if Statements

• An else clause is matched to the last unmatched if (no matter what the indentation implies)

```java
if (age < 21)
    if (age > 2)
        System.out.println("age is larger than" + " 2 and smaller than 21");
    else
        System.out.println("age is " + "smaller than 21");
```

• Braces can be used to specify the if statement to which an else clause belongs.
Quiz

- What is printing the algorithm described in the flowchart below, given 3 numbers (A, B and C) as input:
import java.util.Scanner;

public class MinOfThree
{
    public static void main (String[] args)
    {
        int num1, num2, num3, min = 0;

        Scanner scan = new Scanner (System.in);

        System.out.println ("Enter three integers: ");
        num1 = scan.nextInt();
        num2 = scan.nextInt();
        num3 = scan.nextInt();

        continue
    }
}
continue

if (num1 < num2)
    if (num1 < num3)
        min = num1;
    else  //matches the last unmatched if
        min = num3;
else
    min = num3;

System.out.println("Minimum value: " + min);
}
```java
continue

if (num1 < num2)
    if (num1 < num3)
        min = num1;
    else
        min = num3;
else
    if (num2 < num3)
        min = num2;
    else
        min = num3;

System.out.println("Minimum value: "+min);
```
Quiz

- What happens in the MinOfThree program if two or more of the values are equal? If exactly two of the values are equal, does it matter whether the equal values are lower or higher than the third?
Quiz

• What happens in the MinOfThree program if two or more of the values are equal? If exactly two of the values are equal, does it matter whether the equal values are lower or higher than the third?

If two or more values are equal, the program still prints the lowest value.
Quiz

• What output is produced by the following code fragment given the assumptions below

```java
if (num1 < num2)
    System.out.println("red ");
if ((num1 + 5) < num2)
    System.out.println("white ");
else
    System.out.println("blue ");
System.out.println("yellow ");
```

a) num1=2, num2=10
b) num1=10, num2=2
c) num1=2, num2=2
Quiz

• What output is produced by the following code fragment given the assumptions below

```java
if (num1 < num2)
    System.out.println("red ");
if ((num1 + 5) < num2)
    System.out.println("white ");
else
    System.out.println("blue ");
System.out.println("yellow ");
```

a) num1=2, num2=10 → red white yellow  
b) num1=10, num2=2 → blue yellow  
c) num1=2, num2=2 → blue yellow
Outline

Boolean Expressions
The if Statement
Comparing Data
The while Statement
Iterators
The ArrayList Class
Determining Event Sources
Check Boxes and Radio Buttons
Comparing Data

• When comparing data using boolean expressions, it's important to understand the nuances of certain data types

• Let's examine some key situations:
  – Comparing floating point values for equality
  – Comparing characters
  – Comparing strings (alphabetical order)
  – Comparing object vs. comparing object references
Comparing Float Values

• You should rarely use the equality operator (==) when comparing two floating point values (float or double)

• Two floating point values are equal only if their underlying binary representations match exactly

• Computations often result in slight differences that may be irrelevant

• In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal
Comparing Float Values

- To determine the equality of two floats, use the following technique:

```java
if (Math.abs(f1 - f2) < TOLERANCE)
    System.out.println("Essentially equal");
```

- If the difference between the two floating point values is less than the tolerance, they are considered to be equal.

- The tolerance could be set to any appropriate level, such as 0.000001.
Comparing Characters

- As we've discussed, Java character data is based on the Unicode character set.
- Unicode establishes a particular numeric value for each character, and therefore an ordering.
- We can use relational operators on character data based on this ordering (\(<\), \(>\), \(==\), \(>=\), \(<=\), \(!=\)).
- For example, the character \('+\)' is less \(<\) than the character \('\U0102\)' because it comes before it in the Unicode character set.
- Appendix C provides an overview of Unicode.
Comparing Characters

• In Unicode, the digit characters (0-9) are contiguous and in order

• Likewise, the uppercase letters (A-Z) and lowercase letters (a-z) are contiguous and in order

<table>
<thead>
<tr>
<th>Characters</th>
<th>Unicode Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 9</td>
<td>48 through 57</td>
</tr>
<tr>
<td>A – Z</td>
<td>65 through 90</td>
</tr>
<tr>
<td>a – z</td>
<td>97 through 122</td>
</tr>
</tbody>
</table>
Quiz

Is this snippet syntactically correct? If yes, what is going to be printed?

```java
if ('a' > 'b')
    System.out.println(true);
else
    System.out.println(false);
```
Quiz

Is this snippet syntactically correct? If yes, what is going to be printed?

```java
if ('a' > 'b')
    System.out.println(true);
else
    System.out.println(false);
```

YES it is correct and will print:
false
Comparing Objects

• The == operator can be applied to objects – it returns **true** if the two **references** are **aliases** of each other

• The **equals** method is defined for all objects, but unless **we redefine** it when we write a class, it has the **same semantics** as the == operator

• It has been **redefined** in the **String** class to compare the characters in the two strings

• When you write a class, you can redefine the **equals** method to return true under whatever conditions are appropriate
Comparing Strings

• Remember that in Java a character string is an object

• The `equals` method can be called with strings to determine if two strings contain exactly the same characters in the same order

• The `equals` method returns a boolean result

```java
if (name1.equals(name2))
    System.out.println("Same name");
```
Comparing Strings

• We cannot use the relational operators to compare strings ('<', or ' >')

• The `String` class contains the `compareTo` method for determining if one string comes before another – *but it returns an integer!*

• A call to `name1.compareTo(name2)`
  
  – returns **zero** if `name1` and `name2` are equal (contain the same characters)
  
  – returns a **negative** value if `name1` is less than `name2`
  
  – returns a **positive** value if `name1` is greater than `name2`
Comparing Strings

- Because comparing characters and strings is based on a character set, it is called a lexicographic ordering.

```java
int result = name1.compareTo(name2);
if (result < 0)
    System.out.println (name1 + "comes first");
else
    if (result == 0)
        System.out.println ("Same name");
    else
        System.out.println (name2 + "comes first");
```
Lexicographic Ordering

- Lexicographic ordering is not strictly alphabetical when uppercase and lowercase characters are mixed.

- For example, the string "Great" comes before the string "fantastic" because all of the uppercase letters come before all of the lowercase letters in Unicode.

- Also, short strings come before longer strings with the same prefix (lexicographically).

- Therefore "book" comes before "bookcase".
Quiz

• Write an `compareTo` method for the `Die` class of section 4.2

• A call to `die1.compareTo(die2)`
  – returns zero if `die1` and `die2` are equal (show the same face)
  – returns a negative value if the `faceValue` of `die1` is smaller than the `faceValue` of `die2`
  – returns a positive value in the other cases
Quiz

• Write an `compareTo` method for the `Die` class of section 4.2

• A call to `die1.compareTo(die2)`
  – returns zero if `die1` and `die2` are equal (show the same face)
  – returns a negative value if the `faceValue` of `die1` is smaller than the `faceValue` of `die2`
  – returns a positive value in the other cases

```java
public int compareTo(Die die) {
    return faceValue - die.faceValue;
}
```
Quiz

• Write an `equals` method for the `Die` class of section 4.2
Quiz

• Write an equals method for the Die class of section 4.2

public boolean equals(Die die) {
    if (faceValue == die.faceValue)
        return true;
    return false;
}

public boolean equals(Die die) {
    return (faceValue == die.faceValue);
}
Quiz

• Write an `equals` method for the `Die` class of section 4.2

• Actually the previous solutions are not OK because the parameter of `equals` must be of type `Object`

```java
public boolean equals(Object die) {
    if (die.getClass() == getClass())
        return faceValue == (((Die) die).faceValue);
    return false;
}
```
Outline

Boolean Expressions
The if Statement
Comparing Data
The while Statement
Iterators
The ArrayList Class
Determining Event Sources
Check Boxes and Radio Buttons
Repetition Statements

- *Repetition statements* allow us to execute a statement multiple times

- Often they are referred to as *loops*

- Like conditional statements, they are controlled by boolean expressions

- Java has three kinds of repetition statements: *while*, *do*, and *for* loops

- The *do* and *for* loops are discussed in Chapter 6
The while Statement

• A *while statement* has the following syntax:

```java
while ( condition )
    statement;
```

• If the *condition* is true, the *statement* is executed

• Then the condition is *evaluated again*, and if it is still true, the statement is *executed again*

• The statement is executed repeatedly until the condition becomes false
Logic of a while Loop

condition evaluated

true

statement

false
The while Statement

• An example of a while statement:

```java
int count = 1;
while (count <= 5)
{
    System.out.println (count);
    count++;
}
```

• If the condition of a while loop is false initially, the statement is never executed

• Therefore, the body of a while loop will execute zero or more times
Sentinel Values

• Let's look at some examples of loop processing
• A loop can be used to maintain a running sum
• A sentinel value is a special input value that represents the end of input
• See Average.java
/**
 * Average.java       Author: Lewis/Loftus
 * 
 * Demonstrates the use of a while loop, a sentinel value, and a running sum.
 */

import java.text.DecimalFormat;
import java.util.Scanner;

public class Average
{
    public static void main(String[] args)
    {
        // Computes the average of a set of values entered by the user.
        // The running sum is printed as the numbers are entered.

        int sum = 0, value, count = 0;
        double average;

        Scanner scan = new Scanner(System.in);

        System.out.print("Enter an integer (0 to quit): ");
        value = scan.nextInt();
        continue
continue

while (value != 0) // sentinel value of 0 to terminate loop
{
    count++;

    sum += value;
    System.out.println ("The sum so far is " + sum);

    System.out.print ("Enter an integer (0 to quit): ");
    value = scan.nextInt();
}

continue
continue

    System.out.println ();

    if (count == 0)
        System.out.println ("No values were entered.");
    else
    {
        average = (double)sum / count;

        DecimalFormat fmt = new DecimalFormat ("0.###");
        System.out.println ("The average is " + fmt.format(average));
    }
}
```java
if (count == 0)
    System.out.println("No values were entered.");
else {
    average = (double)sum / count;
    DecimalFormat fmt = new DecimalFormat("0.###");
    System.out.println("The average is " + fmt.format(average));
}
```
Quiz

• Modify the code of the previous example so that 0 can be entered as any other integer, i.e., will not stop the input

• Hint: use another non numeric "sentinel" and use hasNextInt() method of Scanner (it returns true if the input is an integer)
Solution

System.out.print("Enter an integer ("q" to quit): ");
while (scan.hasNextInt())
{
    value = scan.nextInt();
    count++;
    sum += value;
    System.out.println("The sum so far is " + sum);
    System.out.print("Enter an integer ("q" to quit): ");
}

Input Validation

• A loop can also be used for *input validation*, making a program more *robust*

• It's generally a good idea to verify that input is valid (in whatever sense) when possible

• See *WinPercentage.java*
import java.text.NumberFormat;
import java.util.Scanner;

public class WinPercentage
{
    public static void main (String[] args)
    {
        final int NUM_GAMES = 12;
        int won;
        double ratio;

        Scanner scan = new Scanner (System.in);

        System.out.print ("Enter the number of games won (0 to "+ NUM_GAMES + "): ");
        won = scan.nextInt();
        continue
continue

while (won < 0 || won > NUM_GAMES)
{
    System.out.print ("Invalid input. Please reenter: ");
    won = scan.nextInt();
}

ratio = (double)won / NUM_GAMES;

NumberFormat fmt = NumberFormat.getPercentInstance();

System.out.println ();
System.out.println ();
System.out.println ("Winning percentage: " + fmt.format(ratio));
}
}
continue

```java
while(won < 0 || won > NUM_GAMES) {
    System.out.print("Invalid input. Please reenter:");
    won = scan.nextInt();
}
```

```java
ratio = (double)won / NUM_GAMES;
```

```java
NumberFormat fmt = NumberFormat.getPercentInstance();
```

```java
System.out.println();
System.out.println("Winning percentage: "+fmt.format(ratio));
```

Sample Run

```
Enter the number of games won (0 to 12): -5
Invalid input. Please reenter: 13
Invalid input. Please reenter: 7
```

Winning percentage: 58%
What is going on here?

• An example of an "strange" loop:

```java
int count = 1;
while (count <= 25)
{
    System.out.println (count);
    count = count - 1;
}
```

• This loop will continue executing until interrupted (Control-C) or until an underflow error occurs.
Infinite Loops

• The body of a while loop eventually must make the condition false

• If not, it is called an infinite loop, which will execute until the user interrupts the program

• This is a common logical error

• You should always double check the logic of a program to ensure that your loops will terminate normally
Nested Loops

• Similar to nested if statements, loops can be nested as well

• That is, the body of a loop can contain another loop

• For each iteration of the outer loop, the inner loop iterates completely

• See PalindromeTester.java
import java.util.Scanner;

public class PalindromeTester {
    public static void main(String[] args) {
        String str, another = "y";
        int left, right;

        Scanner scan = new Scanner(System.in);

        while (another.equalsIgnoreCase("y")) // allows y or Y
        {
            System.out.println("Enter a potential palindrome:");
            str = scan.nextLine();
            left = 0;
            right = str.length() - 1;

            continue
        }
    }
}
continue

while (str.charAt(left) == str.charAt(right) && left < right)
{
    left++;   
    right--;  
}

System.out.println();

if (left < right)
    System.out.println ("That string is NOT a palindrome.");
else
    System.out.println ("That string IS a palindrome.");

System.out.println();
System.out.print ("Test another palindrome (y/n)? ");
another = scan.nextLine();
}
```java
while (str.charAt(left) == str.charAt(right) && left < right)
{
    left++;
    right--;
}
System.out.println();
if (left < right)
System.out.println("That string is NOT a palindrome.");
else
System.out.println("That string IS a palindrome.");
System.out.println();
System.out.print("Test another palindrome (y/n)? ");
another = scan.nextLine();
}
```
How many times will the string "Here" be printed?

count1 = 1;
while (count1 <= 10)
{
    count2 = 1;
    while (count2 < 20)
    {
        System.out.println ("Here");
        count2++;
    }
    count1++;
}
Quick Check

How many times will the string "Here" be printed?

count1 = 1;
while (count1 <= 10)
{
    count2 = 1;
    while (count2 < 20)
    {
        System.out.println ("Here");
        count2++;
    }
    count1++;
}

10 * 19 = 190
Quick Check

If \( x \) is an int where \( x = 1 \), what will \( x \) be after the following loop terminates?

\[
\text{while} \ (x < 100) \\
\ x *= 2;
\]
Quick Check

If x is an int where x = 1, what will x be after the following loop terminates?

```java
while (x < 100)
    x *= 2;
```

\( x = 128 \)
Iterators

• An *iterator* is an object that allows you to process a collection of items one at a time

• It lets you step through each item in turn and process it as needed

• An iterator has a `hasNext` method that returns true if there is at least one more item to process

• The `next` method returns the next item

• Iterator objects are defined using the *Iterator* interface, which is discussed further in Chapter 7
Iterators

• Several classes in the Java standard class library are iterators

• The Scanner class is an iterator
  – the hasNext method returns true if there is more data to be scanned
  – the next method returns the next scanned token as a string

• The Scanner class also has variations on the hasNext method for specific data types (such as hasNextInt)
Iterators

• The fact that a Scanner is an iterator is particularly helpful when reading input from a file

• Suppose we wanted to read and process a list of URLs stored in a file

• One scanner can be set up to read each line of the input until the end of the file is encountered

• Another scanner can be set up for each URL to process each part of the path

• See URLDissector.java
import java.util.Scanner;
import java.io.*;

public class URLDissector
{
    // Reads urls from a file and prints their path components.
    public static void main (String[] args) throws IOException
    {
        String url;
        Scanner fileScan, urlScan;

        fileScan = new Scanner (new File("urls.inp"));
        continue
// Read and process each line of the file
while (fileScan.hasNext())
{
    url = fileScan.nextLine();
    System.out.println("URL: " + url);

    urlScan = new Scanner(url);
    urlScan.useDelimiter("/");

    // Print each part of the url
    while (urlScan.hasNext())
        System.out.println("   " + urlScan.next());

    System.out.println();
}
}
// Read and process each line of the file
while (fileScan.hasNext()) {
    url = fileScan.nextLine();
    System.out.println("URL: " + url);
    urlScan = new Scanner(url);
    urlScan.useDelimiter("/");
    // Print each part of the url
    while (urlScan.hasNext())
        System.out.println("   " + urlScan.next());
    System.out.println();
}

Sample Run

URL: www.google.com
    www.google.com

URL: www.linux.org/info/gnu.html
    www.linux.org
    info
    gnu.html

URL: thelyric.com/calendar/
    thelyric.com
    calendar

URL: www.cs.vt.edu/undergraduate/about
    www.cs.vt.edu
    undergraduate
    about

URL: youtube.com/watch?v=EHCRImwRGLs
    youtube.com
    watch?v=EHCRImwRGLs
Outline

Boolean Expressions
The if Statement
Comparing Data
The while Statement
Iterators
The ArrayList Class
Determining Event Sources
Check Boxes and Radio Buttons
The ArrayList Class

- An `ArrayList` object stores a list of objects, and is often processed using a loop.

- The `ArrayList` class is part of the `java.util` package.

- You can reference each object in the list using a numeric index.

- An `ArrayList` object grows and shrinks as needed, adjusting its capacity as necessary.
The ArrayList Class

- Index values of an ArrayList begin at 0 (not 1):
  0  "Bashful"
  1  "Sleepy"
  2  "Happy"
  3  "Dopey"
  4  "Doc"

- Elements can be inserted and removed

- The indexes of the elements adjust accordingly
ArrayList Methods

- Some `ArrayList` methods:

  boolean add (E obj)
  void add (int index, E obj)
  Object remove (int index)
  Object get (int index)
  boolean isEmpty()
  int size()
The ArrayList Class

• The type of object stored in the list is established when the `ArrayList` object is created:

```java
ArrayList<String> names = new ArrayList<String>();
ArrayList<Book> list = new ArrayList<Book>();
```

• This makes use of Java `generics`, which provide additional type checking at compile time

• An `ArrayList` object cannot store primitive types, but that's what wrapper classes are for

• See `Beatles.java`
import java.util.ArrayList;

public class Beatles {
    // Stores and modifies a list of band members.
    public static void main (String[] args) {
        ArrayList<String> band = new ArrayList<String>();

        band.add("Paul");
        band.add("Pete");
        band.add("John");
        band.add("George");

        continue
    }
}
System.out.println (band);
int location = band.indexOf ("Pete");
band.remove (location);

System.out.println (band);
System.out.println ("At index 1: " + band.get(1));
band.add (2, "Ringo");

System.out.println ("Size of the band: " + band.size());
int index = 0;
while (index < band.size())
{
    System.out.println (band.get(index));
    index++;
}
}
System.out.println (band);
int location = band.indexOf ("Pete");
band.remove (location);
System.out.println (band);
System.out.println ("At index 1: " + band.get(1));
band.add (2, "Ringo");
System.out.println ("Size of the band: " + band.size());

int index = 0;
while (index < band.size()) {
    System.out.println (band.get(index));
    index++;
}

Output

[Paul, Pete, John, George]
[Paul, John, George]
At index 1: John
Size of the band: 4
Paul
John
Ringo
George
Outline

Boolean Expressions
The if Statement
Comparing Data
The while Statement
Iterators
The ArrayList Class
Determining Event Sources
Check Boxes and Radio Buttons
Determining Event Sources

• Recall that interactive GUIs require establishing a relationship between components and the listeners that respond to component events

• One listener object can be used to listen to two different components

• The source of the event can be determined by using the `getSource` method of the event passed to the listener

• See `LeftRight.java`
• See `LeftRightPanel.java`
import javax.swing.JFrame;

public class LeftRight {
    //-----------------------------------------------------------------
    //  Creates the main program frame.
    //-----------------------------------------------------------------
    public static void main (String[] args) {
        JFrame frame = new JFrame ("Left Right");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(new LeftRightPanel());
        frame.pack();
        frame.setVisible(true);
    }
}
// LeftRight.java
// Demonstrates the use of a single listener for multiple buttons.

import javax.swing.JFrame;

public class LeftRight {
    //_creates the main program frame.
    public static void main(String[] args) {
        JFrame frame = new JFrame("Left Right");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(new LeftRightPanel());
        frame.pack();
        frame.setVisible(true);
    }
}
// Demonstrates the use of one listener for multiple buttons.

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LeftRightPanel extends JPanel
{
    private JButton left, right;
    private JLabel label;
    private JPanel buttonPanel;

    continue
public LeftRightPanel ()
{
    left = new JButton("Left");
    right = new JButton("Right");

    ButtonListener listener = new ButtonListener();
    left.addActionListener(listener);
    right.addActionListener(listener);

    label = new JLabel("Push a button");

    buttonPanel = new JPanel();
    buttonPanel.setPreferredSize(new Dimension(200, 40));
    buttonPanel.setBackground(Color.blue);
    buttonPanel.add(left);
    buttonPanel.add(right);

    setPreferredSize(new Dimension(200, 80));
    setBackground(Color.cyan);
    add(label);
    add(buttonPanel);
}
continue

//****************************************************************************
//  Represents a listener for both buttons.
//****************************************************************************
private class ButtonListener implements ActionListener {
    //---
    //  Determines which button was pressed and sets the label text accordingly.
    //---
    public void actionPerformed (ActionEvent event) {
        if (event.getSource() == left) {
            label.setText("Left");
        } else {
            label.setText("Right");
        }
    }
}
Outline

Boolean Expressions
The if Statement
Comparing Data
The while Statement
Iterators
The ArrayList Class
Determining Event Sources
Check Boxes and Radio Buttons
Check Boxes

• A check box is a button that can be toggled on or off

• It is represented by the JCheckBox class

• Unlike a push button, which generates an action event, a check box generates an ItemEvent whenever it changes state

• The ItemListener interface is used to define item event listeners

• A check box calls the itemStateChanged method of the listener when it is toggled
Check Boxes

• Let's examine a program that uses check boxes to determine the style of a label's text string

• It uses the `Font` class, which embodies a character font's:
  – family name (such as Times or Courier)
  – style (bold, italic, or both)
  – font size

• See `StyleOptions.java`
• See `StyleOptionsPanel.java`
import javax.swing.JFrame;

public class StyleOptions
{
   // Creates and presents the program frame.
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Style Options");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      StyleOptionsPanel panel = new StyleOptionsPanel();
      frame.getContentPane().add (panel);

      frame.pack();
      frame.setVisible(true);
   }
}
import javax.swing.JFrame;

public class StyleOptions {
    
    public static void main (String[] args) {
        JFrame frame = new JFrame("Style Options");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        StyleOptionsPanel panel = new StyleOptionsPanel();
        frame.getContentPane().add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}

// ******************************************************************************
//  StyleOptions.java       Author: Lewis/Loftus
//
//  Demonstrates the use of check boxes.
//  ******************************************************************************

//********************************************************************
//  StyleOptions.java       Author: Lewis/Loftus
//
//  Demonstrates the use of check boxes.
//  ******************************************************************************
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class StyleOptionsPanel extends JPanel {
    private JLabel saying;
    private JCheckBox bold, italic;

    continue
public StyleOptionsPanel()
{
    saying = new JLabel ("Say it with style!");
saying.setFont (new Font ("Helvetica", Font.PLAIN, 36));

    bold = new JCheckBox ("Bold");
    bold.setBackground (Color.cyan);
    italic = new JCheckBox ("Italic");
    italic.setBackground (Color.cyan);

    StyleListener listener = new StyleListener();
    bold.addItemListener (listener);
    italic.addItemListener (listener);

    add (saying);
    add (bold);
    add (italic);

    setBackground (Color.cyan);
    setPreferredSize (new Dimension (300, 100));
}
private class StyleListener implements ItemListener {
    public void itemStateChanged (ItemEvent event) {
        int style = Font.PLAIN;

        if (bold.isSelected())
            style = Font.BOLD;

        if (italic.isSelected())
            style += Font.ITALIC;

        saying.setFont (new Font("Helvetica", style, 36));
    }
}
Radio Buttons

• A group of *radio buttons* represents a set of mutually exclusive options – only one can be selected at any given time.

• When a radio button from a group is selected, the button that is currently "on" in the group is automatically toggled off.

• To define the group of radio buttons that will work together, each radio button is added to a *ButtonGroup* object.

• A radio button generates an action event.
Radio Buttons

• Let's look at a program that uses radio buttons to determine which line of text to display

• See QuoteOptions.java
• See QuoteOptionsPanel.java
import javax.swing.JFrame;

public class QuoteOptions {

    public static void main (String[] args) {
        JFrame frame = new JFrame ("Quote Options");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        QuoteOptionsPanel panel = new QuoteOptionsPanel();
        frame.getContentPane().add (panel);

        frame.pack();
        frame.setVisible (true);
    }
}
import javax.swing.JFrame;

public class QuoteOptions {

    // Creates and presents the program frame.
    public static void main(String[] args) {
        JFrame frame = new JFrame("Quote Options");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        QuoteOptionsPanel panel = new QuoteOptionsPanel();
        frame.getContentPane().add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}
//********************************************************************
// QuoteOptionsPanel.java        Author: Lewis/Loftus
//
// Demonstrates the use of radio buttons.
//********************************************************************

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class QuoteOptionsPanel extends JPanel {

    private JLabel quote;
    private JRadioButton comedy, philosophy, carpentry;
    private String comedyQuote, philosophyQuote, carpentryQuote;

    // Sets up a panel with a label and a set of radio buttons
    // that control its text.

    public QuoteOptionsPanel() {
        comedyQuote = "Take my wife, please.";
        philosophyQuote = "I think, therefore I am.";
        carpentryQuote = "Measure twice. Cut once.";

        quote = new JLabel (comedyQuote);
        quote.setFont (new Font ("Helvetica", Font.BOLD, 24));
    }
}

continue
continue

    comedy = new JRadioButton("Comedy", true);
    comedy.setBackground(Color.green);
    philosophy = new JRadioButton("Philosophy");
    philosophy.setBackground(Color.green);
    carpentry = new JRadioButton("Carpentry");
    carpentry.setBackground(Color.green);

    ButtonGroup group = new ButtonGroup();
    group.add(comedy);
    group.add(philosophy);
    group.add(carpentry);

    QuoteListener listener = new QuoteListener();
    comedy.addActionListener(listener);
    philosophy.addActionListener(listener);
    carpentry.addActionListener(listener);

    add (quote);
    add (comedy);
    add (philosophy);
    add (carpentry);

    setBackground (Color.green);
    setPreferredSize (new Dimension(300, 100));
}
private class QuoteListener implements ActionListener {

    public void actionPerformed(ActionEvent event) {
        Object source = event.getSource();

        if (source == comedy)
            quote.setText(comedyQuote);
        else if (source == philosophy)
            quote.setText(philosophyQuote);
        else
            quote.setText(carpentryQuote);
    }
}
Summary

• Chapter 5 focused on:
  – boolean expressions
  – the if and if-else statements
  – comparing data
  – while loops
  – iterators
  – more drawing techniques
  – more GUI components