Controlling Execution

Controlling execution

- In Java you make choices with execution control statements
- Keywords
  - if-else
  - while
  - do-while
  - for
  - return
  - break
  - switch

True and false

- **All conditional statements use the truth or falsehood of a conditional expression**
  - **Determine the execution path**
  - **e.g., a == b**

- **Note Java does not allow you to use a number as a *boolean***
  - **Allowed it in C and C++ (where truth is nonzero and falsehood is zero)**
  - **e.g., if(a!=0)**

If-else

- **The if-else statement is the most basic way to control program flow**
  - **The *else* is optional**
  - ***Boolean-expression* must produce a *boolean* result**

```
if(Boolean-expression)
  statement
```

or

```
if(Boolean-expression)
  statement
else
  statement
```

Iteration

- **Looping is controlled by *while*, *do-while* and *for*, which are sometimes classified as *iteration statements***
  - **A statement repeats until the controlling *Boolean-expression* evaluates to *false***
- ***While loop***

```
while(Boolean-expression)
  statement
```

```
1  //: control/WhileTest.java
2  // Demonstrates the while loop.
3
4  public class WhileTest {
5    static boolean condition() {
6      boolean result = Math.random() < 0.99;
7      System.out.print(result + ", ");
8      return result;
9    }
10   public static void main(String[] args) {
11     while(condition())
12       System.out.println("Inside 'while'");
13     System.out.println("Exited 'while'");
14   }
15 } /* (Execute to see output) *///:~
16
```

do-while

- **The sole difference between *while* and *do-while***
  - **the statement of the *do-while* always executes at least once, even if the expression evaluates to *false* the first time**
- **In practice, *do-while* is less common than *while***
- ***Do-While loop***

```
do
   statement
while(Boolean-expression);
```

for

- A for loop is perhaps the most commonly used form of iteration
- *for loop*

```
for(initialization; Boolean-expression; step)
  statement
```

- Any of the expressions *initialization*, *Boolean-expression* or *step* can be empty

```
1  //: control/ListCharacters.java
2  // Demonstrates "for" loop by listing
3  // all the lowercase ASCII letters.
4
5  public class ListCharacters {
6    public static void main(String[] args) {
7      for(char c = 0; c < 128; c++)
8        if(Character.isLowerCase(c))
9          System.out.println("value: " + (int)c +
10           " character: " + c);
11   }
12 }
```

```
/* Output:
value: 97 character: a
value: 98 character: b
value: 99 character: c
value: 100 character: d
value: 101 character: e
value: 102 character: f
value: 103 character: g
value: 104 character: h
value: 105 character: i
value: 106 character: j
...
*///:~
```

for (Cont.)

- **Using the comma operator, you can define multiple variables within a for statement**
  - **They must be of the same type**

```
1  //: control/CommaOperator.java
2
3  public class CommaOperator {
4    public static void main(String[] args) {
5      for(int i = 1, j = i + 10; i < 5; i++, j = i * 2) {
6        System.out.println("i = " + i + " j = " + j);
7      }
8    }
9  }
```

```
/* Output:
i = 1 j = 11
i = 2 j = 4
i = 3 j = 6
i = 4 j = 8
*///:~
```

Foreach syntax

- **Java SE5 introduces a new and more succinct for syntax, for use with arrays and containers**
  - **you don't have to create an *int* to count through a sequence of items**
  - **the foreach produces each item for you, automatically**

```
1  //: control/ForEachFloat.java
2  import java.util.*;
3
4  public class ForEachFloat {
5    public static void main(String[] args) {
6      Random rand = new Random(47);
7      float f[] = new float[10];
8      for(int i = 0; i < 10; i++)
9        f[i] = rand.nextFloat();
10     for(float x : f)
11       System.out.println(x);
12   }
13 }
```

Foreach syntax (Cont.)

- **Any method that returns an array is a candidate for use with foreach**
  - **foreach will also work with any object that is *Iterable***

```
1  //: control/ForEachString.java
2
3  public class ForEachString {
4    public static void main(String[] args) {
5      for(char c : "An African Swallow".toCharArray() )
6        System.out.print(c + " ");
7    }
8  }
```

```
/* Output:
A n  A f r i c a n  S w a l l o w
*///:~
```

return

- **Several keywords represent unconditional branching, which simply means that the branch happens without any test**
  - ***return, break, continue***
- **The *return* keyword has two purposes**
  - **It specifies what value a method will return**
  - **it causes the current method to exit, returning that value**
- **If you do not have a *return* statement in a method that returns *void*, there's an implicit return at the end of that method**
  - **it's not always necessary to include a *return* statement**

break and continue

- **Control the flow of the loop inside the body of any of the iteration statements by using *break* and *continue***
  - ***break*** **quits the loop without executing the rest of the statements in the loop**
  - ***continue*** **stops the execution of the current iteration and goes back to the beginning of the loop to begin the next iteration**

break and continue -- the infamous “goto”

- Although *goto* is a reserved word in Java, it is not used in the language
  - Java has no *goto*
- A label is an identifier followed by a colon, like this: `label1:`
- The *only* place a label is useful in Java is right before an iteration statement
  - the only reason to use labels in Java is when you have nested loops and you want to *break* or *continue* through more than one nested level

```
label1:
outer-iteration {
  inner-iteration {
    //...
    break; // (1)
    //...
    continue; // (2)
    //...
    continue label1; // (3)
    //...
    break label1; // (4)
  }
}
```

The infamous “goto” (Cont.)

```java
//: control/LabeledWhile.java
// While loops with "labeled break" and "labeled continue."
import static net.mindview.util.Print.*;

public class LabeledWhile {
  public static void main(String[] args) {
    int i = 0;
    outer:
    while(true) {
      print("Outer while loop");
      while(true) {
        i++;
        print("i = " + i);
        if(i == 1) {
          print("continue");
          continue;
        }
        if(i == 3) {
          print("continue outer");
          continue outer;
        }
        if(i == 5) {
          print("break");
          break;
        }
        if(i == 7) {
          print("break outer");
          break outer;
        }
      }
    }
  }
}
```

```
/* Output:
Outer while loop
i = 1
continue
i = 2
i = 3
continue outer
Outer while loop
i = 4
i = 5
break
Outer while loop
i = 6
i = 7
break outer
*///:~
```

The infamous “goto” (Cont.)

- **The rules:**
  - **A plain *continue* goes to the top of the innermost loop and continues.**
  - **A labeled *continue* goes to the label and reenters the loop right after that label.**
  - **A *break* “drops out of the bottom” of the loop.**
  - **A labeled *break* drops out of the bottom of the end of the loop denoted by the label.**
- **The only reason to use labels in Java is when you have nested loops and you want to *break* or *continue* through more than one nested level**

switch

- **selection statement**

```
switch(integral-selector) {
  case integral-value1 : statement; break;
  case integral-value2 : statement; break;
  case integral-value3 : statement; break;
  case integral-value4 : statement; break;
  case integral-value5 : statement; break;
  // ...
  default: statement;
}
```

switch (Cont.)

- **Example**

```
1  //: control/VowelsAndConsonants.java
2  // Demonstrates the switch statement.
3  import java.util.*;
4  import static net.mindview.util.Print.*;
5
6  public class VowelsAndConsonants {
7    public static void main(String[] args) {
8      Random rand = new Random(47);
9      for(int i = 0; i < 100; i++) {
10       int c = rand.nextInt(26) + 'a';
11       printnb((char)c + ", " + c + ": ");
12       switch(c) {
13         case 'a':
14         case 'e':
15         case 'i':
16         case 'o':
17         case 'u': print("vowel");
18                   break;
19         case 'y':
20         case 'w': print("Sometimes a vowel");
21                   break;
22         default:  print("consonant");
23       }
24     }
25   }
26 }
```

Thank you

zhenling@seu.edu.cn