If Statements

- The syntax of a simple if statement is as follows:

```
if (booleanExpression)
  trueBody
else
  falseBody
```

- booleanExpression is a boolean expression and trueBody and falseBody are each either a single statement or a block of statements enclosed in braces ("{" and "}").
Compound if Statements

- There is also a way to group a number of boolean tests, as follows:

```java
if (firstBooleanExpression)
    firstBody
else if (secondBooleanExpression)
    secondBody
else
    thirdBody
```

Switch Statements

- Java provides for multiple-value control flow using the switch statement.
- The switch statement evaluates an integer, string, or enum expression and causes control flow to jump to the code location labeled with the value of this expression.
- If there is no matching label, then control flow jumps to the location labeled “default.”
- This is the only explicit jump performed by the switch statement, however, so flow of control “falls through” to the next case if the code for a case is not ended with a `break` statement.
Switch Example

```java
switch (d) {
    case MON:
        System.out.println("This is tough.");
        break;
    case TUE:
        System.out.println("This is getting better.");
        break;
    case WED:
        System.out.println("Half way there.");
        break;
    case THU:
        System.out.println("I can see the light.");
        break;
    case FRI:
        System.out.println("Now we are talking.");
        break;
    default:
        System.out.println("Day off!");
}
```

Break and Continue

- Java supports a `break` statement that immediately terminate a while or for loop when executed within its body.

- Java also supports a `continue` statement that causes the current iteration of a loop body to stop, but with subsequent passes of the loop proceeding as expected.
While Loops

- The simplest kind of loop in Java is a `while` loop.
- Such a loop tests that a certain condition is satisfied and will perform the body of the loop each time this condition is evaluated to be true.
- The syntax for such a conditional test before a loop body is executed is as follows:

  ```java
  while (booleanExpression)
  {
    loopBody
  }
  ```

Do-While Loops

- Java has another form of the while loop that allows the boolean condition to be checked at the end of each pass of the loop rather than before each pass.
- This form is known as a do-while loop, and has syntax shown below:

  ```java
  do
  {
    loopBody
  }
  while (booleanExpression)
  ```
For Loops

- The traditional for-loop syntax consists of four sections—an initialization, a boolean condition, an increment statement, and the body—although any of those can be empty.
- The structure is as follows:

```java
for (initialization; booleanCondition; increment)
    loopBody
```
- Meaning:

```java
{
    initialization;
    while (booleanCondition) {
        loopBody;
        increment;
    }
}
```

Example For Loops

- Compute the sum of an array of doubles:

```java
public static double sum(double[] data) {
    double total = 0;
    for (int j=0; j < data.length; j++) // note the use of length
        total += data[j];
    return total;
}
```

- Compute the maximum in an array of doubles:

```java
public static double max(double[] data) {
    double currentMax = data[0]; // assume first is biggest (for now)
    for (int j=1; j < data.length; j++) // consider all other entries
        if (data[j] > currentMax) // if data[j] is biggest thus far...
            currentMax = data[j]; // record it as the current max
    return currentMax;
}
```
For-Each Loops

- Since looping through elements of a collection is such a common construct, Java provides a shorthand notation for such loops, called the **for-each** loop.
- The syntax for such a loop is as follows:
  
  ```java
  for (elementType name : container)
      loopBody
  ```

For-Each Loop Example

- Computing a sum of an array of doubles:
  
  ```java
  public static double sum(double[ ] data) {
      double total = 0;
      for (double val : data) // Java’s for-each loop style
          total += val;
      return total;
  }
  ```
- When using a for-each loop, there is no explicit use of array indices.
- The loop variable represents one particular element of the array.
Simple Output

- Java provides a built-in static object, called `System.out`, that performs output to the "standard output" device, with the following methods:
  
  ```java
  print(String s): Print the string s.
  print(Object o): Print the object o using its toString method.
  print(baseType b): Print the base type value b.
  println(String s): Print the string s, followed by the newline character.
  println(Object o): Similar to print(o), followed by the newline character.
  println(baseType b): Similar to print(b), followed by the newline character.
  ```

Simple Input

- There is also a special object, `System.in`, for performing input from the Java console window.
- A simple way of reading input with this object is to use it to create a `Scanner` object, using the expression
  ```java
  new Scanner(System.in)
  ```
- Example:
  ```java
  import java.util.Scanner;  // loads Scanner definition for our use
  
  public class InputExample {
      public static void main(String[] args) {
          Scanner input = new Scanner(System.in);
          System.out.print("Enter your age in years: ");
          double age = input.nextDouble();
          System.out.print("Enter your maximum heart rate: ");
          double rate = input.nextDouble();
          double fb = (rate - age) * 0.65;
          System.out.println("Your ideal fat-burning heart rate is "+ fb);
      }
  }
  ```
The Scanner class reads the input stream and divides it into tokens, which are strings of characters separated by delimiters.

- `hasNext()`: Return `true` if there is another token in the input stream.
- `next()`: Return the next token string in the input stream; generate an error if there are no more tokens left.
- `hasNextType()`: Return `true` if there is another token in the input stream and it can be interpreted as the corresponding base type, `Type`, where `Type` can be `Boolean`, `Byte`, `Double`, `Float`, `Int`, `Long`, or `Short`.
- `nextType()`: Return the next token in the input stream, returned as the base type corresponding to `Type`; generate an error if there are no more tokens left or if the next token cannot be interpreted as a base type corresponding to `Type`.

Sample Program

```java
public class CreditCard {
    // Instance variables:
    private String customer; // name of the customer (e.g., "John Bowman")
    private String bank; // name of the bank (e.g., "California Savings")
    private String account; // account identifier (e.g., "5391 0375 9387 5309")
    private int limit; // credit limit (measured in dollars)
    protected double balance; // current balance (measured in dollars)

    // Constructors:
    public CreditCard(String cust, String bk, String acct, int lim, double initialBal) {
        customer = cust;
        bank = bk;
        account = acct;
        limit = lim;
        balance = initialBal;
    }
    public CreditCard(String cust, String bk, String acct, int lim) {
        this(cust, bk, acct, lim, 0.0); // use a balance of zero as default
    }
}
```

© 2014 Goodrich, Tamassia, Goldwasser
Sample Program

```java
// Accessor methods:
public String getCustomer() { return customer; }
public String getBank() { return bank; }
public String getAccount() { return account; }
public int getLimit() { return limit; }
public double getBalance() { return balance; }

// Update methods:
public boolean charge(double price) { // make a charge
    if (price + balance > limit) // if charge would surpass limit
        return false; // refuse the charge
    // at this point, the charge is successful
    balance += price; // update the balance
    return true; // announce the good news
}
public void makePayment(double amount) { // make a payment
    balance -= amount;
}

// Utility method to print a card's information
public static void printSummary(CreditCard card) {
    System.out.println("Customer = " + card.customer);
    System.out.println("Bank = " + card.bank);
    System.out.println("Account = " + card.account);
    System.out.println("Balance = " + card.balance); // implicit cast
    System.out.println("Limit = " + card.limit); // implicit cast
}

// main method shown on next page...
```

---

Sample Program

```java
public static void main(String[] args) {
    CreditCard[] wallet = new CreditCard[3];
    wallet[0] = new CreditCard("John Bowman", "California Savings",
         5391 0375 9387 5309, 5000);
         3485 0399 9396 1954, 3500);
         5391 0375 9387 5309, 2500, 300);

    for (int val = 1; val <= 16; val++) {
        wallet[0].charge(3+val);
        wallet[1].charge(2+val);
        wallet[2].charge(val);
    }

    for (CreditCard card : wallet) {
        card.printSummary(card); // calling static method
        while (card.getBalance() > 200.0) {
            card.makePayment(200);
        }
        System.out.println("New balance = " + card.getBalance());
    }
}
```

© 2014 Goodrich, Tamassia, Goldwasser