CMP-4008Y Programming I

Geoff McKeown - ***Lecture Notes Week 3***

Conditional Statements. Switch statements. Java's Parameter-Passing Mechanism

Lecture Objectives

- To introduce two of Java's conditional statements.
- Using named constants in Java
- To introduce Java's `switch` statement.
- To discuss Java's parameter passing mechanism.
- To distinguish between instance variables and local variables.
- To review how methods are invoked in Java.

The `if` and `if-else` statements

- One important way in which boolean expressions are used in Java is in
  - `if`
  - `if-else`

  conditional statements.
- An `if` statement has the form

```
if ( <some condition> )
    <if condition body>
```

Example 1

Suppose `n` and `remainder` are `int` variables and `isDivisibleBy5` is a `boolean` variable

```
remainder = n % 5;
if (remainder == 0)
    isDivisibleBy5 = true;
```

Example 2

```
if ( age >= 18 )
    System.out.println("You may vote");
```

◇ If the ***<if condition body>*** involves more than one statement then these must be enclosed between opening and closing curly braces ({ and }).

Suppose `n`, `remainder` and `factor` are `int` variables,and `isDivisibleBy5` is a `boolean` variable

```
remainder = n % 5;
if (remainder == 0){
    isDivisibleBy5 = true;
    factor = n / 5;
}
```

We might extend our second example as follows:

```
if ( age >= 18 ){
    System.out.println("You may vote");
    System.out.println("You may also legally buy beer!");
}
```

- An `if`-`else` statement has the form

```
if ( some condition )
    <if condition body>
else
    <else condition body>
```

Example 1

```
remainder = n % 5;
if (remainder == 0)
    isDivisibleBy5 = true;
else
    isDivisibleBy5 = false;
```

Example 2

```
if ( age >= 18 )
 System.out.println("You may vote");
else
     System.out.println("You will have to wait" + (18-age) +
                 " years before you may vote");
```

- Again if either the *<if condition body>* or the *<else condition body>*involves more than one statement then these must be enclosed between opening and closing curly braces ({ and }).

Example 1

Suppose `dividend` is another `int` variable

```
remainder = n % 5;
if (remainder == 0){
    isDivisibleBy5 = true;
    factor = n / 5;
}
else{
    isDivisibleBy5 = false;
    dividend = n / 5;
}
```

Example 2

```
if ( age >= 18 ){
    System.out.println("You may vote");
    System.out.println("You may also legally buy beer!");
}
else{
     System.out.println("You will have to wait" + (18-age)
                + " years before you may vote");
     System.out.println("Some say lemonade is fine");
}
```

```
import java.util.Scanner;
/*
 * A Java application to demonstrate the use of a conditional statement
 * Author Geoff McKeown
 */
public class AgeCheck
{
    public static void main( String [ ] args )
    {
        Scanner scan = new Scanner(System.in);

        int age;
        System.out.println("Enter your age as a  whole number");
        age = scan.nextInt();

        if ( age >= 18 ){
            System.out.println("You may vote");
            System.out.println("You may also legally buy beer!");
        }
        else{
             System.out.println("You will have to wait" + (18-age)
                        + " years before you may vote");
             System.out.println("Some say lemonade is fine");
        System.out.println( "Finished" );
    }
}
```

Named constants - the `final` modifier

- A *constant* represents permanent data that never changes.
- It is good practice to use names for constants in Java programs to help make the code self-documenting;
- In Java, we use the `final` modifier to indicate that an identifier represents a value that cannot be changed:

```
final double POUNDS_IN_KILOS = 0.454;
```

- By convention, upper-case letters are used for the names of constants: if we want to use more than one word in the name, we separate the words by an underscore character.
- In one of your previous lab sheets, you had to write a Java program to convert a number representing a value in pounds to the equivalent value in kilograms.
- Most of you used the *literal* value 0.454 directly in your programs but it is better style to use a named constant:

```
/*
 * A program to convert a number representing a value in pounds
 * to the equivalent value in kilograms
 */
package poundstokilos;
import java.util.Scanner;

public class PoundsToKilos {

    public static void main(String[] args) {

    final double POUNDS_IN_KILOS = 0.454;


    Scanner scan = new Scanner(System.in);
    System.out.println("Enter an amount in pounds: ");
    double pounds = scan.nextDouble();


    double kilos = POUNDS_IN_KILOS * pounds;


    System.out.println("The amount " + pounds + " in pounds is equivalent to "
        +"the amount " + kilos + " in kilograms");
    }
}
```

switch statements

- As an alternative to using multiple `if` or nested `if` statements to select an execution path from a number of possible execution paths, a `switch` statement may sometimes be used.
- A *switch multiple selection statement* performs different actions for each possible value of a *switch-expression* that must evaluate to give a value of type `byte`, `short`, `int` or `char`.
- In addition to these four primitive types, a switch-expression can also have type `String`, and it also works with a value of an enumerated type (see a later lecture).
- A switch statement has the general form:

```
switch (<switch-expression>) {
    case <value 1>:  <action 1>;
            break;
    case <value 2>:  <action 2>;
            break;
        .   .   .
    case <value n>:  <action n>;
            break;
default:  <default action>;
}
```

Example 1

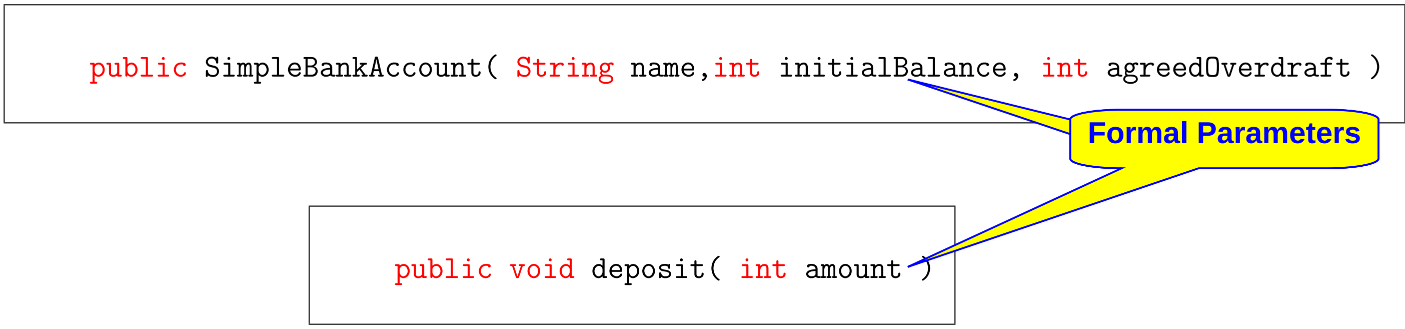
```
import java.util.Random;
/*
 * SwitchDemo.java: To demonstrate the use of a switch statement
 * and the use of the Random class which
 * provides methods for generating random numbers.
 */
public class SwitchDemo {
    public static void main(String[] args) {

        // define four int constants

        final int HEARTS = 0;
        final int DIAMONDS = 1;
        final int CLUBS = 2;
        final int SPADES = 3;

        Random randomNumbers = new Random();
        int  cardSuit;

        // select a card at random and output its suit
        cardSuit = randomNumbers.nextInt(4);
                        // random number from 0 to 3
            switch ( cardSuit ){
                case HEARTS:    System.out.println("A heart was drawn.");
                                break;
                case DIAMONDS:  System.out.println("A diamond was drawn.");
                                break;

                case CLUBS:     System.out.println("A club was drawn.");
                                break;
                case SPADES:    System.out.println("A spade was drawn.");
                                break;
            }
    }
}
```

Example 2

```
/*
 * To demonstrate the use of Strings in a switch statement.
 * Author: Geoff McKeown
 */
package stringswitchdemo;
import java.util.Scanner;

public class StringSwitchDemo {
    public static void main(String[] args) {

       // define seven String constants
        final String MONDAY_STR
            = "Monday is the first day of week";
        final String TUESDAY_STR
            = "Tuesday is the second day of week";
        final String WEDNESDAY_STR
            = "Wednesday is the third day of week";
        final String THURSDAY_STR
            = "Thursday is the fourth day of the week";
        final String FRIDAY_STR
            = "Friday is the fifth day of the week";
        final String SATURDAY_STR
            = "Saturday is the first day of the weekend";
        final String SUNDAY_STR
            = "Sunday is the second day of the weekend";

        // enter data from the keyboard
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter a day of the week\n");
        String day = scan.next().toLowerCase();

        switch (day) {
            case "monday":
                System.out.println(MONDAY_STR);
                break;
            case "tuesday":
                System.out.println(TUESDAY_STR);
                break;
            case "wednesday":
                System.out.println(WEDNESDAY_STR);
                break;
            case "thursday":
                System.out.println(THURSDAY_STR);
                break;

            case "friday":
                System.out.println(FRIDAY_STR);
                break;
            case "saturday":
                System.out.println(SATURDAY_STR);
                break;
            case "sunday":
                System.out.println(SUNDAY_STR);
                break;
            default :
                System.out.println("Invalid day entered");
        }
    }
}
```

Parameters

- Java uses a *parameter passing mechanism* to provide any additional information needed by a constructor or a method.
- The name and type of each required parameter is specified in the header:



```
public SimpleBankAccount( String name,int initialBalance, int agreedOverdraft )
```

```
public void deposit( int amount )
```

- Parameters specified in a header are called *formal parameters*.
- A formal parameter is available to an object only within the body of the constructor or method in which it is declared.
- Computer memory is allocated to a formal parameter only when the constructor or method in which it is declared is executed.
- When the constructor (or method) is invoked, an *actual parameter* is given corresponding to each formal parameter.

- When the constructor (or method) executes, each actual parameter is stored in the space allocated to the corresponding formal parameter.

- When an object is created, memory is allocated to an *instance variable* corresponding to each of its fields.
- An instance variable is a *variable*:
  - a variable is a *name* (*identifier*) associated with a portion of space in main memory suitable for holding a value of a particular type;
  - for example, the memory associated with a variable of type `int` stores (a representation of) a whole number.
- An object is created from a class by invoking a constructor using the `new` operator.

```
    SimpleBankAccount myAccount =
               new SimpleBankAccount("Geoff", 5000, 3000);
```

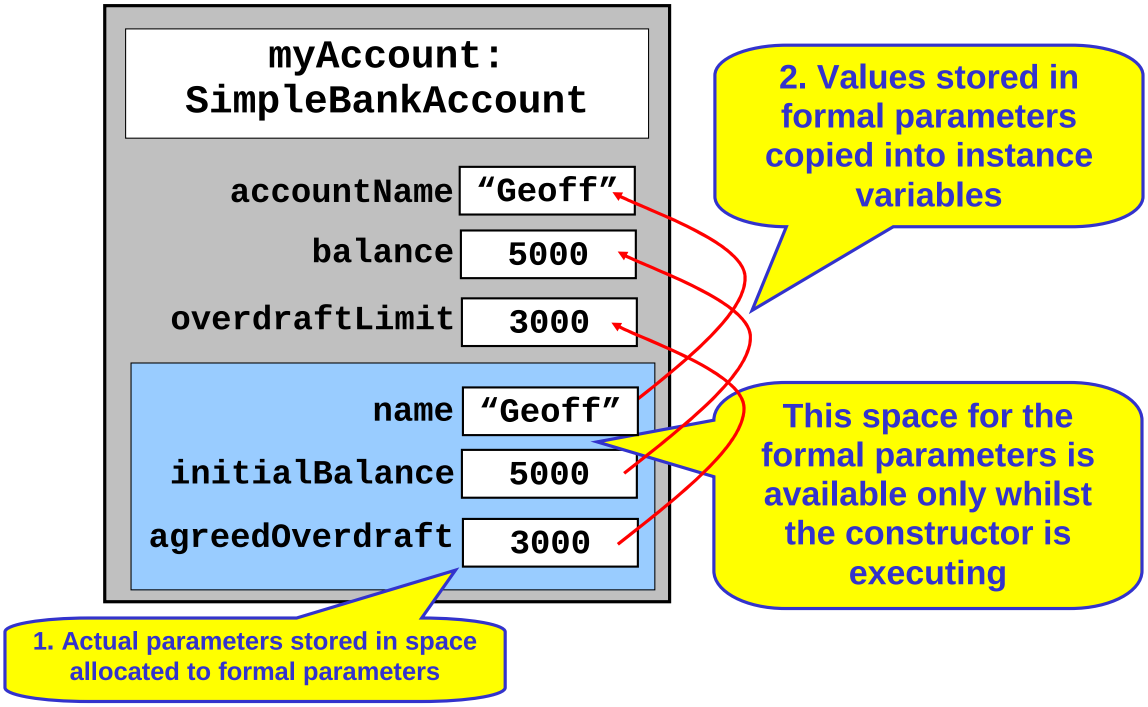
These are the actual parameters

- This causes the constructor

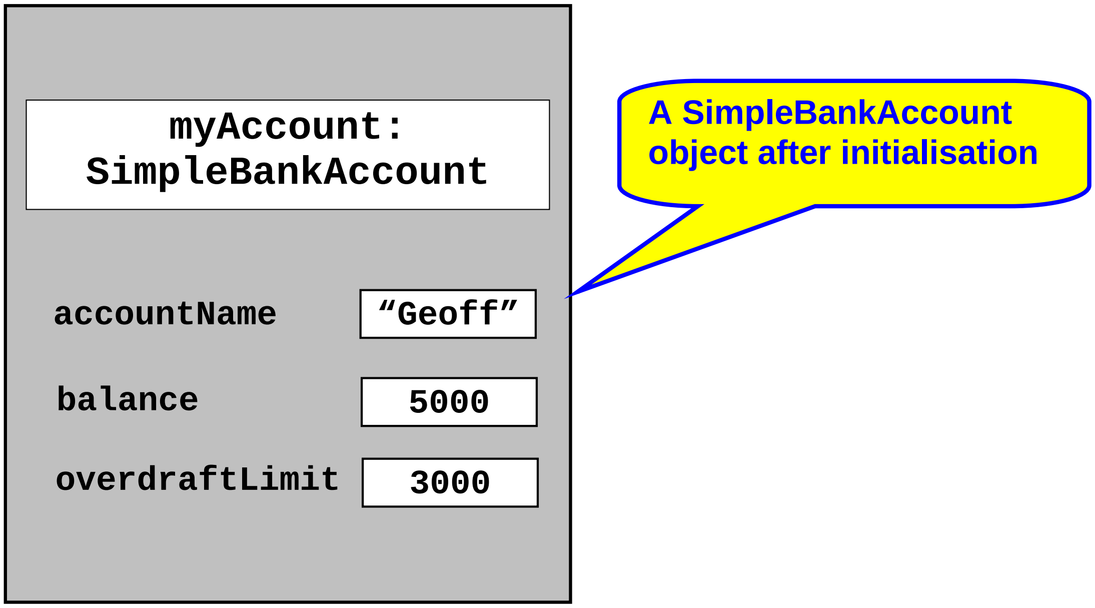
```
          SimpleBankAccount(name, initialBalance, agreedOverdraft);
```

  to be executed. The process is illustrated as follows:

- The values of an object's instance variables (fields) define the object's state.
- Constructors store initial values into the instance variables.
- Each class instance has its own state.


myAccount:
SimpleBankAccount
accountName
“Geoff”
balance
5000
overdraftLimit
3000
name
“Geoff”
initialBalance
5000
agreedOverdraft
3000
2. Values stored in formal parameters copied into instance variables
This space for the formal parameters is available only whilst the constructor is executing
1. Actual parameters stored in space allocated to formal parameters

- ◇ Instance variables (corresponding to fields) are used to store data persistent throughout an object’s existence;
  - ▷ that is, the *lifetime* of an instance variable is identical to the lifetime of the object.
  - ▷ because their lifetime corresponds to that of the object in which they are defined, and they are accessible throughout the whole class, they are said to have *class scope*.



Local variables

- A *local variable* is declared and used within a single method or constructor.
- Its declaration does not contain a visibility modifier.
- Recall the `main` method in the class `SimpleBankAccountDriver`

```
import java.util.Scanner;
/**
 * A class to run SimpleBankAccount
 * Author Geoff McKeown
 */
public class SimpleBankAccountDriver
{
    public static void main( String [ ] args )
    {
        String name = "Geoff";
        int initialBalance;
        int overDlimit;

        .   .   .

    }
}
```

Local variable definitions.

- A local variable can only be used inside the *block*
  - i.e. the sequence of statements enclosed between a pair of curly braces ({ and })

  in which it is declared.
- The lifetime of a *local variable* ends after the execution of the last statement in the block in which it is declared.

Invoking Methods

- All objects of a given class can invoke the same set of methods as defined in that class.
- To get an object to invoke a method in Java, we use the *"dot operator"*:

```
objectName.methodName(<actual parameter list>)
```

- Each item in the *<actual parameter list>* consists of a value, or a variable to which a value has been assigned, of the same type as the corresponding formal parameter in the *<formal parameter list>*. Recall that the latter is given in the header of the method definition in the body of the class.

```
public class SimpleBankAccountDriver
{
   public static void main( String [ ] args )
   {
      String name = "Geoff";
      int initialBalance;
      int overDlimit;

      Scanner scan = new Scanner(System.in);

      System.out.print("Enter a value (a whole number)"
                   + " for the opening balance:  " );
      initialBalance = scan.nextInt();

      System.out.print("Enter a value (a whole number) "
                   + "for the agreed overdraft limit:  " );
      overDlimit = scan.nextInt();
      SimpleBankAccount myAccount =
         new SimpleBankAccount(name, initialBalance, overDlimit);

      System.out.println("The current balance of account "
          + myAccount + " is " + myAccount.getBalance() );

      myAccount.deposit(100);
      System.out.println("The current balance of account "
          + myAccount + " is now " + myAccount.getBalance() );

      System.out.println("The current overdraft limit of account "
          + myAccount + " is " + myAccount.getOverdraftLim() );

      System.out.println( "Finished" );
   }
}
```

Method invocations

Method invocations