CSC 142

Classes and methods revisited

[Reading: chapters 6 and 14]

Overloading: an example

- Printing in a console window

```
System.out.print("Hello"); // a string
System.out.print(true); // a boolean
System.out.print('c'); // a character
System.out.print(3); // an integer
System.out.print(new Double(5.1));//an object
```

- In the class PrintStream (class type of out), there are several methods with the same name, namely print.
  They differ by their formal parameter list.

Method overloading

- In a class, you can define multiple methods with the same name.
- The compiler picks the right method based on the arguments passed to the method.
- Advantage: give the illusion that one method works on many types.
- Compiler gives an error if there is more than one possible match. If the match is not exact, the compiler does some automatic conversion.
- Complete matching algorithm rather complex

Example: multiple constructors

```
public class MyClass
{
   public MyClass(){/* default */}
   public MyClass(int i){/* code */}
   public MyClass(double x){/* code */}
   public MyClass(String s, int i){/* code */}
   public MyClass(Integer i){/*code*/}
}
// for a user of MyClass
MyClass c1=new MyClass();
MyClass c2=new MyClass(3);
MyClass c3=new MyClass(3.4);
MyClass c4=new MyClass("Hello",34);
MyClass c5=new MyClass(new Double(3)); //Nope!
```

Another example (1)

- Define an area method in a Calculator class.
  - pass to area an object (type Square, Circle, Triangle…) and return its area. These types are defined by some other classes.

-
  ```
   Calculator m = new Calculator();
  //Default square,circle and rectangle
  Square s = new Square();
  Circle c = new Circle();
  Triangle t = new Triangle();
  // Areas of s,c and t
  System.out.println("s: "+m.area(s));
  System.out.println("c: "+m.area(c));
  System.out.println("t: "+m.area(t));
  ```

Another example(2)

- Write 3 versions of area in class Calculator

```
public double area(Square s)
{/* Return the area of Square s */}
public double area(Triangle t)
{/* Return the area of Triangle t */}
public double area(Circle c)
{/* Return the area of Circle c */}
```

- See the full code on the class web site
- Design issue:
  - Should the area functionality be in Calculator, or in each of the classes Square, Circle,…?

Ambiguity

```
public class SomeClass{
   public void foo(double x,int i)
   {/*version 1*/}
    public void foo(int i,double x)
    {/*version 2*/}
 }
 // for a user of SomeClass
 SomeClass c = new SomeClass();
 c.foo(4.5,3); // version 1
 c.foo(5,4.5); // version 2
 c.foo(1,2); // which one?
```

- The compiler will tell you.

static method (1)

- In a Calendar class, define a method to check if a year is a leap year

```
 public boolean isALeapYear(int year){
  return ((year%4==0 && year%100!=0) ||
          year%400==0);
 }
```

- By design, isALeapYear doesn't use any instance of Calendar. It uses only the formal parameter year.
- Don't declare it as an instance method. Declare it as a class method

```
public static boolean isALeapYear(int year)
```

- To call the method, write e.g.

```
 if (Calendar.isALeapYear(y))...
```

static method (2)

- In a static method, only other static methods, constants or variables of the class can be used.
- A non static member of a class is always associated with an object (this). this is not available in a static method since the method can be called without an instance of the class.

public static void main(String[] args)

- All Java applications start by executing a special method called main
  - In BlueJ, no need for it since BlueJ allows the user to manipulate the objects interactively.
- In other environments (e.g. JBuilder), write:

```
 public class MyClass{
    public static void main(String[] args){
     MyClass c = new MyClass();
     c.foo(); // some method in MyClass
    }
 }
```

What about String[] args?

- When executing a java program,
  - arguments can be provided to the program via the command line
  - Write 'java *className* parm1 parm2 …' (e.g. in a DOS window).
  - parm1, parm2, … are stored in the String array args
- Example: Salutations.java (available as a sample program).
  - interactive mode: enter the name using a dialog box
  - command line mode: write 'java Salutations Valerie'

Local variables: scope and lifetime

- Local variables: variables defined within methods (formal parameters are local variables)
- Scope (where a variable can be used):
  - From the declaration line to the brace closing the block where the declaration line is.
- Lifetime (How long a variable is available)
  - A local variable is created when the control flow is on its declaration line. It is destroyed when leaving the block where the variable is declared

Example

```
public void foo(int i)
//i is created and initialized with the actual
//parameter value every time foo is executed
{
  // print the digits of i one by one
  do{
     int j; //created at every iteration
            //of the loop
     j=i%10;
     System.out.println(j);
     i/=10;
  }while(i>0);//j is destroyed
  // create and initialize s
  String s="Thanks for using foo";
  System.out.println(s);
 }// s and i are destroyed
```

Object: scope and lifetime

- An object can be used wherever a reference to the object is available.
- Visibility of the instance fields:
  - everywhere if declared public within the class of the object
  - only within the class of the object if declared private
- Lifetime of an object
  - Created by the call to the class constructor (using new)
  - Destroyed when all references to the object are destroyed (the memory is reclaimed by the garbage collector).

Example

```
public class Drawing{
  //the window reference and a GWindow object are
  //created whenever a Drawing is constructed
  private GWindow window = new GWindow();
  public Drawing()
  {
   Rectangle r = new Rectangle();
   window.add(r);
  }// r is deleted. What about the Rectangle it
   // referred to? Can't tell. But may be kept
   // if the GWindow object creates a reference
   // to it in the add method.
}

// in some other method: the Drawing is kept
// as long there is a reference to access it.
Drawing d = new Drawing();
```

What about class fields?

- Created when the class is loaded by the JVM
- Destroyed when the application terminates
- Visibility: same as for instance fields
  - public: everywhere
  - private: only within the class

```
public class MyClass{
   // count available everywhere
   // (as MyClass.count) as soon as MyClass
   // is loaded by the JVM.
   public static int count;
   // more code…
 }
```

Packages in Java (1)

- A package is a named collection of classes. It defines a namespace for the classes that it contains.
- For large projects, packages prevent name collision.
- e.g., if you create a JOptionPane class, the compiler won't confuse it with the usual JOptionPane class, which is actually the javax.swing.JOptionPane class.

Packages in Java (2)

- To create a package use the package keyword.

```
package org.seattlecentral;
// import statements
public class MyClass{/*code*/}
```

- MyClass is actually org.seattlecentral.MyClass
- On the computer, the java compiler creates the bytecode file MyClass.class in the folder seattlecentral inside the folder org, which is in the current directory.
- For our projects, we don't use packages. All classes are then in a default unnamed package.

Program organization

- Write each public class in its own java file
    - public class Date is written in the file Date.java
- Can have more than one class in a java file. But only one of them is public

```
public class Class1{ /* code */}
class Class2 {/* code */}
```

- Class1 and Class2 are in the same package. If no package is specified, that would be the default package. But, only Class1 is visible to another class importing the package.

More on scope

- An instance field may be declared without any access modifier (public or private). Its visibility is restricted to the package of the class.
- Inside a package, any class has access to the data and method members of other classes as long as the members are not declared private.
- But only public members of public classes are visible outside the package.

```
public class Class1{
  int i; /* package access */
  private int j; /* Class1 only */
  public int k; /* everybody */ }
class Class2 {
  public int a; /* package access */
  int b;  /* package access */
  private int c; /* Class2 only */}
```