Parameters and Objects

Chapter 3

A Class Metaphor

Classes, like blueprints, describe

An Object Metaphor

Objects, like houses, are usable

Attributes and Behaviors

Features or members

- Attributes
  - Characterize or distinguish an object
  - Are a data values (variables) held by objects
  - Each object maintains a private copy of each attribute value

- Behaviors
  - Are what an object can do (the services or operations provided by an object)
  - Are implemented as methods
  - Methods are called through or bound to an object, called "this object"
  - This object is the default target of the method (i.e., the data in this object is accessed by name only)

Access Control Modifiers

Control access to a class’s features

- private: class scope; methods and instance variables

- default (aka friendly): class and package scope; methods, instance variables, and classes

- protected: class, package, and subclass scope; methods and instance variables

- public: global scope; methods, instance variables, and classes

Methods

The active part of a class

```
int a = 100;
int b = 25;

50 10  
\[ a \]

a+50 2\[ b \]
\[ x \]
\[ y \]
\[ x + y \]

5 4 3
```

- When used or called, a method can be treated as a black box
  - User cares only about what the method does, not about how (performance and errors aside)
  - The method’s interface or signature is a description of how to connect with or call the method

- Method input (parameters) may be
  - Constants
  - Variables
  - Expressions
**Method Execution**

Call and return sequences

```
main( )
 {   method1( )
     {   method2( );
         return;
     }
     method1( );
 }   return;

method3( );
 return;
```

**Method Definition Syntax**

AKA declaration syntax

- Always defined inside a class but can not be nested
- Header
  - Modifiers: 0 or more; common: public, private, etc.
  - Return value type: requested, may be void
  - Name: Java style is "Camel notation"
  - Argument list: formal parameters – (‘ and ’) are always required
  - Definitions require typing info: documentation shown in this format
- Body
  - Between '{' and '}'
  - Contains any number of legal Java statements (including 0)

```
public class Employee
{
    public Manager boss = new Employee();
    public static int getBss(int depart, int when) { ... }  // non-static
    public static int getPhoneNumber() { ... }  // static
}
```

**Method Call Syntax**

AKA invocation, dispatch

- **Does not include modifiers or typing information**
- Actual parameters may be expressions (constants, variables, method calls, or arithmetic expressions using any of the 3)
- Non-static methods are called through an object
- Static methods are called through the class

```
Employee worker = new Employee();
Manager boss = worker.getBss(416, 2009 + 47);  // non-static
int number = Employee.getPhoneNumber();  // static
```

**Method Return Values**

Getting a value from a method

- Methods terminate by "falling off the end" or calling return
  - return may be called anywhere in the method
  - Method return values are specified with the return statement (i.e., using return is the only way to get a value returned from a method to the calling scope)
  - A method that does not return a value has a "return" type of void
  - Return values may be ignored

**Method Overloading**

Methods with the same name

- Overloaded methods
  - Defined in the same class
  - Have the same name
  - Must have different signatures
  - May have different return types but cannot overload on return type
  - Should perform similar tasks
- Arguments
  - println(char)
  - println(int)
  - println(double)
  - println(String)

```
signature

```

**Constructors**

A special-purpose method

```
public class Point
{
    int x, y;
    public Point( )
    {   x = 100;
        y = 200;
    }
    public Point(int px, int py)
    {   x = px;
        y = py;
    }
    Point p1 = new Point( );
    Point p2 = new Point(50, 25);

    p1 = 100, 200
    p2 = 50, 25
```

- Have the same name as the class
- No return type or value
- May be complex or may be simple
- May be overloaded
- **Default constructors do not have parameters**

```
```
### Value Vs. Reference

**Accessing data**
- Primitive data are accessed by value
  - \( x + y \)
  - `int x = 10; int y = 20;`
  - `x = 10; y = 20`
- Value stored in `x + value stored in y`
- `Circle c; // define reference var`
- `c = new Circle(10,0,0); // instantiation only`
- `Circle c = new Circle(10,0,0); // both in 1 statement`

### The this Reference

**Object self reference**
- "Secret" argument passed to each non-static method
- Points to the calling object; binds it's data to the method
- Used for self reference

```java
class Epi {
  double r, a;
  void draw() {
    x = f1(a);
    y = f2(r,a);
  }
}

class Spoo {
  Epi e1 = new Epi(100, 17);
  Epi e2 = new Epi(17, 18);
  Epi e3 = new Epi(47, 39);
  void display() {
    e2.draw();
  }
}
```

### Packages

**Organizing libraries**
- Java organizes related classes into a "package"
- A Java file can state to which package it belongs with the `package` keyword
- Each package occupies a separate subdirectory
- Package names must be the same as the name of the directory in which the compiled (i.e., bytecode) files reside
- Each compiled class occupies a separate file in the package subdirectory
- Programs access the contents of a package by importing
  - `import packagename.classname;`
  - `import packagename.*;`

### Scope

**Where an identifier (symbol) is visible**

```java
public void method() {
  int r; // local variable
  ... 
  int k; // k goes out of scope
  ... 
  if (r goes out of scope)
  }
}
```

### Object Assignment and Reference

**Creating object aliases**
- Object names are references that "point" to an object
- Multiple names (aliases) may reference the same object
  - Created by assignment
  - Created by method calls

```java
Circle c = new Circle(100,20,10);
Circle t = c; // does not duplicate
Circle t = new Circle(); // method call
public void draw(Circle c) {
  ...
}
```

### Pass By Value: Simple Types

**Illustrated with int**

```java
int x = 100;
inc(x);
```

```java
void inc(int a) {
  a += 100;
}
define x call inc run inc return
time
```
**Pass By Value: Objects**

Assume class Employee

```java
Employee x = new Employee("Fred");
update(x);
void update(Emp a) {
    a = new Employee("Sid");
}
```

**Pass By Value: Updating Objects**

Class Employee has an instance variable name

```java
Employee x = new Employee("Fred");
update(x);
void update(Emp a) {
    a.name = "Sid";
}
```

**Strings Revisited**

Two views

- Can be treated as an atomic entity (chap 2, slide 8)
- Can be treated as a list or string of characters
  - Individual characters accessed by position (i.e., index or subscript)
  - Legal index values range from 0 to length - 1
  - char charAt(int index)
  - char[] toCharArray()
  - String substring(int beginIndex)
  - String substring(int beginIndex, int endIndex)
  - Throws an IndexOutOfBoundsException if index < 0 or index >= length

```java
String s = "Hello";
```

**Exceptions**

Dealing with runtime errors

- Claiming an exception
  - throws Introduces a comma-separated list of exceptions that a method might throw

- Throwing an exception
  - throw Launch an exception object to indicate that an exception situation has occurred

- Catching an exception
  - try Define a block of code that anticipates exceptions
  - catch Define a block of code that deals with a specific exception
  - finally Define a block of code that executes regardless of whether an exception is generated or not

**Input with the Scanner Class**

Details and examples in java.util.Scanner

- Some Constructors
  - Scanner(InputStream source)
  - Scanner(String source)

- Some Methods
  - double nextDouble()
  - int nextInt()
  - String nextLine()
  - boolean hasNextDouble()
  - boolean hasNextInt()
  - boolean hasNextLine()

```java
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
while (sc.hasNextLine()) {
    String line = sc.nextLine();
    // process line
}
```

**Input with showInputDialog**

Pause for input; returns a String

```java
String input = JOptionPane.showInputDialog(null, 
    "Please enter a number of pennies", 
    "Money Input", JOptionPane.QUESTION_MESSAGE);
int pennies = Integer.parseInt(input);
```