CS 112  Introduction to Programming

User-Defined Data Types

Yang (Richard) Yang
Computer Science Department
Yale University
308A Watson, Phone: 432-6400
Email: yry@cs.yale.edu
Admin

- PS6 (Sukoku) questions?
- Planning of remaining of the semester
any program you might want to write

objects
methods and classes
graphics, sound, and image I/O
arrays
conditionals and loops
Math
text I/O
primitive data types
assignment statements
Exercise: Gene Finding

Pre-genomics era. Sequence a human genome.
Post-genomics era. Analyze the data and understand structure.

Genomics. Represent genome as a string over \{ A, C, T, G \} alphabet.

Gene. A substring of genome that represents a functional unit.
- Preceded by ATG. [start codon]
- Multiple of 3 nucleotides. [codons other than start/stop]
- Succeeded by TAG, TAA, or TGA. [stop codons]

Goal. Find all genes.

```
ATAGATGCGCATAGCTAGATG

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
```
public class GeneFind {
    // Q: What about return an array of genes
    public static void findGenes(String start,
        String stop,
        String genome) {
        int beg = -1;
        for (int i = 0; i < genome.length() - 2; i++) {
            String codon = genome.substring(i, i+3);
            if (codon.equals(start)) beg = i;
            if (codon.equals(stop) && beg != -1) { //??
                String gene = genome.substring(beg+3, i);
                if (gene.length() % 3 == 0) {
                    System.out.println(gene);
                    beg = -1;
                }
            }
        } // end of for
    } // end of findGenes
} // end of class
Method and class are programming organization concepts

- Method provides an abstraction to group a sequence of statements
  - abstraction: we can use it without knowing its details

- Class
  - So far our usage is to group similar methods (such as a folder) together, e.g., the Math class pre-defined by Java
  - A general, important function of class is to organize both data and behaviors
Road Ahead: Object Oriented Programming

- Four steps
  - Struct data type
  - Encapsulation
  - Class inheritance
  - Polymorphism
A Programming Problem

- Given a file of cities' \((x, y)\) coordinates, which begins with the number of cities:

  6
  50 20
  90 60
  10 72
  74 98
  5 136
  150 91

- Write a program to draw the cities and then drop a center point and turn all cities red that are within a given radius:

  Center site \(x\)? 100  
  Center site \(y\)? 100  
  Radius? 75
A Less Elegant Solution

Scanner input = new Scanner(new File("cities.txt"));
int cityCount = input.nextInt();
int[] xCoords = new int[cityCount];
int[] yCoords = new int[cityCount];

Scanner console = new Scanner( System.in );
int xCenter = console.nextInt();
int yCenter = console.nextInt();

for (int i = 0; i < cityCount; i++) {
    xCoords[i] = input.nextInt();
    yCoords[i] = input.nextInt();
    if (distance(xCoords[i], yCoords[i], xCenter, yCenter) < THRESHOLD) {
        // draw red
    } else …
}

- **Parallel arrays**: 2 arrays with related data at same indexes.
- **Conceptually**, we should have just a single array of points.
A Conceptual Design

Scanner input = new Scanner(new File("cities.txt"));
int cityCount = input.nextInt();
Point[] points = new Point[cityCount];
Point pointCenter = readPointFromUser();

for (int i = 0; i < cityCount; i++) {
    points[i] = readPoint(input);
    if (distance(points[i], pointCenter) < THRESHOLD) {
        // draw red
    }
}
...

- By introducing the concept of Point, we abstract away the details of the coordinate system
  - Even if later the program changes to another coordinate system, this part of code does not need to change.
Summary: Complex Data Type

Motivation

Scanner input = new Scanner(new File("cities.txt"));
int cityCount = input.nextInt();
int[] xCoords = new int[cityCount];
int[] yCoords = new int[cityCount];
for (int i = 0; i < cityCount; i++) {
xCoords[i] = input.nextInt();
yCoords[i] = input.nextInt();
if (distance(xCoords[i], yCoords[i], xCenter, yCenter) < THRESHOLD) {
    // draw red
}
}

Scanner input = new Scanner(new File("cities.txt"));
int cityCount = input.nextInt();
Point[] points = new Point[cityCount];
for (int i = 0; i < cityCount; i++) {
    points[i] = readPoint(input);
    if (distance(points[i], pointCenter) < THRESHOLD) {
        // draw red
    }
}
Language Support: Defining Point

- Need a language structure to specify that each point is defined by two data components:
  - The x coordinate
  - The y coordinate

```java
public class Point {
    int x;
    int y;
}
```

- Save this code into a file named `Point.java`.

- The above code creates a **new user-defined type (class)** named `Point`.
- The `Point` class definition serves as a **template** for creating `Point` **objects**.
public class Point {
    int x;
    int y;
}

- **x or y is called a field**
  - A non-static, class scope variable defined inside a class (new type)
  - A field is also called an instance variable, or an attribute
  - A field is part of the descriptive characteristics (state) of an object
  - Each object has *its own copy* of each field

- **More example:**

```java
public class BankAccount {
    String acctName;
    int acctNumber;
    double balance;
}
```
A Class and its client

- Point.java is not, by itself, a runnable program.
  - A class can be used by multiple client programs.

**PointMain1.java (client program)**
```java
public class PointMain1 {
    public static void main(String args) {
        Point p1 = new Point();
        p1.x = 50;
        p1.y = 27;

        Point p2 = new Point();
        p2.x = 98;
        p2.y = 53;
    }
}
```

**Point.java (class of objects)**
```java
public class Point {
    int x;
    int y;
}
```

```
x 50  y 27
```

```
x 98  y 53
```
public class PointMain2 {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point();
        p1.x = 50; p1.y = 27;

        Point p2 = new Point();
        p2.x = 98; p2.y = 53;

        System.out.println(p1.x + "", " + p1.y); // 50, 27
        System.out.println(p2.x + "", " + p2.y); // 98, 53

        // move p2
        p2.x += 2;
        p2.y ++;
        System.out.println(p2.x + "", " + p2.y); // 100, 55
    }
}
Summary of Important Concepts: Class, Object, Variable, Field

```java
public class Point {
    int x;
    int y;
}
```

- The Point class definition serves as a template for creating Point objects.
- A field (also called instance variable, attribute) is a class-scoped, non-static variable describing characteristics of each object. Each object has its own copy of each field.
- A Point variable stores a reference to a Point object.
Summary: **Class, Object, Variable, Field**

- **Object variables (references)**
  - `PointMain.java` (client program)
  - `public class PointMain {
      main(String args) {
          Point p1 = new Point();
          p1.x = 50;
          p1.y = 27;
          Point p2 = p1;
          Point p3 = new Point();
          p3.x = 98;
          p3.y = 53;
          ...
      }
  }

- **objects**
  - `Point.java` (class of objects)
  - `public class Point {
      int x;
      int y;
  }

- **fields**
  - `x` 50
  - `y` 27
  - `x` 98
  - `y` 53

- **class**
  - ` PointMain.java (client program)
  - `public class PointMain {
      main(String args) {
          Point p1 = new Point();
          p1.x = 50;
          p1.y = 27;
          Point p2 = p1;
          Point p3 = new Point();
          p3.x = 98;
          p3.y = 53;
          ...
      }
  }

  - `Point.java (class of objects)
  - `public class Point {
      int x;
      int y;
  }

  - `x` 50
  - `y` 27
  - `x` 98
  - `y` 53`
Historical Note

- Before object-oriented programming, traditional languages (e.g. C) already have a language structure called `struct` to allow user to define data type (combine multiple data fields).

- Object-oriented languages go one step further to organize both data and methods together.
Object Oriented Programming Overview

- OOP philosophy. Software is a simulation of the real world.
  - We know (approximately) how the real world works.
  - Design software to model the real world.

- Procedural programming. [verb-oriented]
  - Tell the computer to do this.
  - Tell the computer to do that.

- Objected oriented programming (OOP). [noun-oriented]
  - Programming paradigm based on data types.
  - Identify objects that are part of the problem domain or solution.
  - Identity: objects are distinguished from other objects (references).
  - State: objects know things (instance variables).
  - Behavior: objects do things (methods).
Object Oriented Programming

"Object-oriented programming is an exceptionally bad idea which could only have originated in California."
-- Edsger Dijkstra
Alan Kay

- Alan Kay. [Xerox PARC 1970s]
  - Invented Smalltalk programming language.
  - Conceived Dynabook portable computer.
  - Ideas led to: laptop, modern GUI, OOP.

“THE COMPUTER REVOLUTION HASN'T STARTED YET.”

“THE BEST WAY TO PREDICT THE FUTURE IS TO INVENT IT.”

“If you don't fail at least 90 per cent of the time, you're not aiming high enough.”

— Alan Kay

Alan Kay
2003 Turing Award
Motivation: Drawing Points

- Suppose our client program wants to draw `Point` objects:

```java
// draw each city
Point p1 = new Point();
p1.x = 15;
p1.y = 37;

StdDraw.filledCircle(p1.x, p1.y, 3);
StdDraw.textLeft(p1.x, p1.y,
                 "(" + p1.x + ", " + p1.y + ")")
```

- To draw other points, the same code must be repeated.
  - We can remove this redundancy using a method.
We can eliminate the redundancy with a static method:

```java
// Draws the given point using StdDraw
public static void draw(Point p) {
    StdDraw.filledCircle(p.x, p.y, 3);
    StdDraw.textLeft(p.x, p.y,
        "(" + p.x + ", " + p.y + ")" );
}
```

`main` would call the method as follows:
```
draw(p1);
```

Question: where do we define the method `draw(Point p)`?
Where do we Define `draw(p)`:

**Attempt 1**: in each client e.g., `PointMain1`
Where do we Define draw(p):
Attempt 2: in Point as a Static Method

```java
public class Point {
    int x;
    int y;

    public static void draw(Point p) {
        StdDraw.filledCircle(p.x, p.y, 3);
        StdDraw.textLeft(p.x, p.y,
                         "(" + p.x + ", " + p.y + ")");
    }
}

Point p1 = new Point();
p1.x = 7; p1.y = 2;
Point.draw(p1);

Point p2 = new Point();
p2.x = 4; p2.y = 3;
Point.draw(p2);
```
Consistent Data Access and Method Access

Point p = new Point();
p.x = 10;
p.y = 5;
Point.draw(p);

p.draw();
Instance Methods

- **instance method (or object method):** Exists inside each object of a class and gives behavior to each object.

  ```java
  public type name(parameters) {
    statements;
  }
  ```

  - same syntax as static methods, but without static keyword

  **Example:**

  ```java
  public void shout() {
    System.out.println("HELLO THERE");
  }
  ```
Instance Method example

Instance method (or object method) can access all class scope variables

```java
public class Point {
    int x;
    int y;

    // Draws this Point object with the given pen.
    public void draw() {
        StdDraw.filledCircle(x, y, 3);
        StdDraw.textLeft(x, y, "(" + x + ", " + y + ")");
    }
}
```

- The `draw` method no longer has a `Point p` parameter.
- How will the method know which point’s x/y to draw?

<table>
<thead>
<tr>
<th>p1: Point</th>
<th>p2: Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 50</td>
<td>x = 98</td>
</tr>
<tr>
<td>y = 27</td>
<td>y = 53</td>
</tr>
</tbody>
</table>
Invoking Instance Method

- Format
  `<obj>.<instance method>(...)`

- The `<obj>` provides an `implicit parameter`
The Implicit Parameter

- During the call `p1.draw();`
  the object referred to by `p1` is the *implicit* parameter.

- During the call `p2.draw();`
  the object referred to by `p2` is the *implicit* parameter.

- In an instance method, when we refer to a field, we are referring to the field of the implicit parameter.
  - We say that it executes in the *context* of a particular object.
Summary: Defining Related Method and Data in the Same Class: Instance Method

```java
public class Point {
    int x;
    int y;

    public static void draw(Point p) {
        StdDraw.filledCircle(p.x, p.y, 3);
        StdDraw.textLeft(p.x, p.y, "(" + p.x + ", " + p.y + ")");
    }
}
```

Point p1 = new Point();
p1.x = 7; p1.y = 2;
p1.draw(); // Point.draw(p1);

Point p2 = new Point();
p2.x = 4; p2.y = 3;
p2.draw(); // Point.draw(p2);

p1 provides the implicit parameter: The x and y in draw() are those of the object referenced by p1.

p2 provides the implicit parameter: The x and y in draw() are those of the object referenced by p2.
Each `Point` object has its own copy of the `draw` method, which operates on that object's state:

```java
Point p1 = new Point();
p1.x = 7;
p1.y = 2;

Point p2 = new Point();
p2.x = 4;
p2.y = 3;

p1.draw();
p2.draw();
```

```java
public void draw() {
    // this code can see p1's x and y
}
```

```java
x 7 y 2
```

```java
public void draw() {
    // this code can see p2's x and y
}
```

```java
x 4 y 3
```
Static Method vs Instance Method

public class Point {
    int x;
    int y;
    public static void draw(Point p) {
        StdDraw.filledCircle(p.x, p.y, 3, 3);
        StdDraw.textLeft(p.x, p.y, "(" + p.x + ", " + p.y + ")");
    }
}

public class Point {
    int x;
    int y;
    public static void draw(Point p) {
        StdDraw.filledCircle(p.x, p.y, 3, 3);
        StdDraw.textLeft(p.x, p.y, "(" + p.x + ", " + p.y + ")");
    }
}

The two approaches differ mostly in syntax. Important: static method cannot access instance variables.
Benefit of Instance Method

```
Point p = new Point();
p.x = 10;
p.y = 5;
p.draw();
```

**Consistent** syntax of accessing both data and behaviors: a class is an abstraction for a collection of objects with common

- data fields (attributes) and
- behaviors/services (i.e., what such objects can do or be done to them)
Instance Method questions

- Write an instance method `toString` to generate string \((x, y)\)

- Write an instance method `translate` that changes a `Point`'s location by a given \(dx, dy\) amount.

- Write an instance method `distanceFromOrigin` that returns the distance between a `Point` and the origin, \((0, 0)\).

Use the formula: 
\[
\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}
\]
public class Point {
    int x;
    int y;

    public void translate(int dx, int dy) {
        x = x + dx;
        y = y + dy;
    }
    public double abs() {
        return Math.sqrt(x * x + y * y);
    }
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
    public void draw() {
        StdDraw.filledCircle(x, y, 3);
        StdDraw.textLeft(x, y, toString());
    }
}
Initializing objects

- Currently it takes 3 lines to create a `Point` and initialize it:

  ```java
  Point p = new Point();
p.x = 3;
p.y = 8;                          // tedious
  ```

- We'd rather specify the fields' initial values at the start:

  ```java
  Point p = new Point(3, 8);  // better!
  ```

  - We are able to do this with most types of objects in Java.
Constructors

- **constructor**: a special method to initialize the state of new objects.

  ```java
  public type(parameters) {
    statements;
  }
  ```

  - runs when the client uses the `new` keyword
  - no return type should be specified; it implicitly "returns" the new object being created

  - If a class has no constructor, Java gives it a *default constructor* with no parameters that sets all fields to zero-equivalent values.
 Constructor example

```java
public class Point {
    int x;
    int y;

    // Constructs a Point at the given x/y location.
    public Point(int initialX, int initialY) {
        x = initialX;
        y = initialY;
    }

    public void translate(int dx, int dy) {
        x = x + dx;
        y = y + dy;
    }

    ...
}
```
public class PointMain3 {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point(5, 2);
        Point p2 = new Point(4, 3);

        // print each point
        System.out.println("p1: (" + p1.x + ", " + p1.y + ")");
        System.out.println("p2: (" + p2.x + ", " + p2.y + ")");

        // move p2 and then print it again
        p2.translate(2, 4);
        System.out.println("p2: (" + p2.x + ", " + p2.y + ")");
    }
}

OUTPUT:
p1: (5, 2)
p2: (4, 3)
p2: (6, 7)
Multiple Constructors

- A class can have multiple constructors.
  - Each one must accept a unique set of parameters (same rule of method overloading).

- Exercise: Write a `Point` constructor with no parameters that initializes the point to (0, 0).
Multiple Constructors

- A default Point constructor with no parameters that initializes the point to (0, 0).

```java
// Constructs a new point at (0, 0).
public Point() {
    x = 0;
    y = 0;
}
```
Common Constructor Issues

1. By accidentally giving the constructor a return type, it is actually not a constructor, but a method named `Point`

```java
public void Point(int initialX, int initialY) {
    x = initialX;
    y = initialY;
}
```
2. Declare a local variable with the same name as a field. The field is "shadowed" by the local variable. Rather than storing value into the field, the param is passed to local variable. The field remains 0.

```java
public class Point {
    int x;
    int y;

    public Point(int initialX, int initialY) {
        int x = initialX;
        int y = initialY;
    }
    ...
}
```
Shadowing

- **shadowing**: 2 variables with same name in same scope.
  - Normally illegal, except when one variable is a field

```java
public class Point {
    int x;
    int y;
    ...
    // this is legal
    public Point(int x, int y) {
        System.out.println("x = " + x);// para x
    }
}
```

- In most of the class, x and y refer to the fields.
- In `Point(int x, int y)`, x and y refer to the method's parameters.
The *this* keyword

- **this**: Refers to the implicit parameter inside your class.

  *(a variable that stores the object on which a method is called)*

- Refer to a field: `this.field`

- Call a method: `this.method(parameters)`
Fixing Shadowing

- To refer to the data field $x$, say `this.x`
- To refer to the parameter $x$, say $x$

```java
public class Point {
    int x;
    int y;
    ...
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public void setLocation(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```
public class Point {
    private int x;
    private int y;

    public Point() {
        this(0, 0); // calls (x, y) constructor
    }

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    ...
}

- Avoids redundancy between constructors
- Only a constructor (not a method) can call another constructor
Summary: Class Definition Components

- **Variables**
  - fields (instance variables per object)
  - static variables (shared by all objects)

- **Methods**
  - static methods (method usable with or without object)
  - instance methods (can be used only on objects)
    - Constructors
    - Accessors (do not change object state)
    - Mutators (modify object state)