CS 112 Introduction to Programming
(Spring 2012)

Lecture #21: Designing Data Types

Zhong Shao

Department of Computer Science
Yale University
Office: 314 Watson

http://flint.cs.yale.edu/cs112

Acknowledgements: some slides used in this class are taken directly or adapted from those accompanying the textbook: Introduction to Programming in Java: An Interdisciplinary Approach by Robert Sedgewick and Kevin Wayne (Copyright 2002-2010)
Object Oriented Programming

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

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.

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. [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
Encapsulation
Encapsulation

Data type. Set of values and operations on those values.
Ex. int, String, Complex, Vector, Document, GuitarString, ...

Encapsulated data type. Hide internal representation of data type.

Separate implementation from design specification.
- Class provides data representation and code for operations.
- Client uses data type as black box.
- API specifies contract between client and class.

Bottom line. You don't need to know how a data type is implemented in order to use it.
Intuition

Client

API
- volume
- change channel
- adjust picture
- decode NTSC signal

Implementation
- cathode ray tube
- electron gun
- Sony Wega 36XBR250
- 241 pounds

client needs to know how to use API

implementation needs to know what API to implement

Implementation and client need to agree on API ahead of time.
Can substitute better implementation without changing the client.
Counter Data Type

Counter. Data type to count electronic votes.

```java
public class Counter {
    public int count;
    public final String name;

    public Counter(String id) { name = id; }
    public void increment() { count++;
    public int value() { return count;

}
```

Legal Java client.

```java
Counter c = new Counter("Volusia County");
c.count = -16022;
```

Oops. Al Gore receives -16,022 votes in Volusia County, Florida.
Counter Data Type

Counter. Encapsulated data type to count electronic votes.

```java
public class Counter {
    private int count;
    private final String name;

    public Counter(String id) { name = id; }
    public void increment() { count++; }
    public int value() { return count; }
}
```

Does not compile.

Counter c = new Counter("Volusia County");
c.count = -16022;

Benefit. Can guarantee that each data type value remains in a consistent state.
Changing Internal Representation

Encapsulation.
- Keep data representation hidden with **private** access modifier.
- Expose API to clients using **public** access modifier.

```java
public class Complex {
    private final double re, im;

    public Complex(double re, double im) {
    }
    public double abs() {
    }
    public Complex plus(Complex b) {
    }
    public Complex times(Complex b) {
    }
    public String toString() {
    }
}
```

**Advantage.** Can switch internal representation without changing client.

**Note.** All our data types are already encapsulated!
Time Bombs

Internal representation changes.


Lesson. By exposing data representation to client, might need to sift through millions of lines of code in client to update.
Ask, Don't Touch

Encapsulated data types.
- Don't touch data and do whatever you want.
- Instead, ask object to manipulate its data.

"Ask, don't touch."

Adele Goldberg
Former president of ACM
Co-developed Smalltalk

Lesson. Limiting scope makes programs easier to maintain and understand.

"principle of least privilege"
Immutability
**Immutability**

**Immutable data type.** Object's value cannot change once constructed.

<table>
<thead>
<tr>
<th>mutable</th>
<th>immutable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Picture</td>
<td>Charge</td>
</tr>
<tr>
<td>Histogram</td>
<td>Color</td>
</tr>
<tr>
<td>Turtle</td>
<td>Stopwatch</td>
</tr>
<tr>
<td>StockAccount</td>
<td>Complex</td>
</tr>
<tr>
<td>Counter</td>
<td>String</td>
</tr>
<tr>
<td>Java arrays</td>
<td>primitive types</td>
</tr>
</tbody>
</table>
Immutability: Advantages and Disadvantages

Immutable data type. Object's value cannot change once constructed.

Advantages.
- Avoid aliasing bugs.
- Makes program easier to debug.
- Limits scope of code that can change values.
- Pass objects around without worrying about modification.

Disadvantage. New object must be created for every value.
Final Access Modifier

Final. Declaring an instance variable to be `final` means that you can assign it a value only once, in initializer or constructor.

```java
public class Counter {
    private final String name;
    private int count;
    ...
}
```

Advantages.

- Helps enforce immutability.
- Prevents accidental changes.
- Makes program easier to debug.
- Documents that the value cannot change.

This value doesn't change once the object is constructed.

This value changes by invoking instance method.
Spatial Vectors
Vector Data Type

**Set of values.** Sequence of real numbers. [Cartesian coordinates]

### API.

```java
public class Vector {
    // create a vector with the given Cartesian coordinates
    Vector(double[] a) {
    }

    // sum of this vector and b
    Vector plus(Vector b) {
    }

    // difference of this vector and b
    Vector minus(Vector b) {
    }

    // scalar product of this vector and t
    Vector times(double t) {
    }

    // dot product of this vector and b
    double dot(Vector b) {
    }

    // magnitude of this vector
    double magnitude() {
    }

    // unit vector with same direction as this vector
    Vector direction() {
    }
}
```

**Example:**

```
x = (0, 3, 4, 0),  y = (0, -3, 1, -4)
x + y = (0, 0, 5, -4)
3x = (0, 9, 12, 0)
x \cdot y = (0 \cdot 0) + (3 \cdot -3) + (4 \cdot 1) + (0 \cdot -4) = -5
|x| = (0^2 + 3^2 + 4^2 + 0^2)^{1/2} = 5
\vec{x} = x / |x| = (0, 0.6, 0.8, 0)
```
Vector Data Type Applications

Relevance. A quintessential mathematical abstraction.

Applications.
- Statistics.
- Linear algebra.
- Clustering and similarity search.
- Force, velocity, acceleration, momentum, torque.
- ...

Vector Data Type: Implementation

public class Vector {
    private int N;
    private double[] coords; // instance variables

    public Vector(double[] a) {
        N = a.length;
        coords = new double[N];
        for (int i = 0; i < N; i++)
            coords[i] = a[i];
    } // constructor

    public double dot(Vector b) {
        double sum = 0.0;
        for (int i = 0; i < N; i++)
            sum += (coords[i] * b.coords[i]);
        return sum;
    }

    public Vector plus(Vector b) {
        double[] c = new double[N];
        for (int i = 0; i < N; i++)
            c[i] = coords[i] + b.coords[i];
        return new Vector(c); // methods
    }
}
This. The keyword \texttt{this} is a reference to the invoking object.

Ex. When you invoke \texttt{a.magnitude()}, \texttt{this} is an alias for \texttt{a}.

```java
public Vector times(double t) {
    double[] c = new double[N];
    for (int i = 0; i < N; i++)
        c[i] = t * coords[i];
    return new Vector(c);
}

public double magnitude() {
    return Math.sqrt(this.dot(this));
}

public Vector direction() {
    return this.times(1.0 / this.magnitude());
}
...