Imperative Programming
Paradigm

Procedural Programming

Imperative/Procedural Paradigm

• The imperative paradigm is characterized by the finite state machine computational model.
• The imperative programming paradigm assumes that the program maintains a modifiable store.
• By changing the values of variables we alter what is stored, thus record state changes.
• Computations are performed through a sequence of steps specified by a list of commands.
• When imperative programming is combined with subprograms, it is called procedural programming.
Procedural Programming

- The model does not scale well and large programs that use this model are difficult to maintain.
- It is difficult to find which functions are affected when the structure of the shared store is changed in any way.
- It is difficult to find relationships between functions that communicate indirectly through the shared store.

Modules

- One solution is to divide a program into units of separately compiled components called modules.
- Most procedural languages have some mechanism for creating modules.
- Each module has its own store which is shared internally.
- A module is encapsulated by a wall through which identifiers are invisible unless explicitly declared part of the modules interface.
- Modules are less powerful than abstract data types because they cannot be instantiated.
Object-Oriented Programming

Topics

• Object-oriented thinking
  – Agents, messages, responsibility
• Abstract data types
• Inheritance
• Polymorphism
• Late binding

Object-Oriented Thinking

• Problem: How to get flowers to Grandma in California?
• Procedural solution:
  – 1. Plant flowers
  – 2. Water flowers
  – 3. Pick flowers
  – 4. Drive to CA
• Object-oriented solution:
  – 1. Call local florist
Agents and Responsibilities

- Me
  - Local Florist
    - CA Florist
      - Delivery Person
      - Flower Arranger
      - Wholesaler
    - Grandma
    - Grower

Narrowing the Semantic Gap

- The power of metaphor
  - Thinking about problems in terms of slots and values doesn’t provide much insight into how a program should be structured.
  - Thinking in terms of behaviors and responsibilities of agents, brings to mind a wealth of intuition, ideas, and understanding from everyday experience.

- Computation as simulation
  - Define a set of entities and how they interact, then set the system in motion.

- Programming by delegation
  - Object-oriented programmers think first about passing the buck.

Responsibility and Independence

*Ask not what you can do to your data structures, but what your data structures can do for you.*

- Timothy Budd
Abstract Data Types

• **Quiz:** What is the IEEE754 standard 32-bit representation of the floating point decimal number -5.125?

• When (for example) a float is incremented, its representation is changed. How is it possible for a programmer to accomplish the change without knowing the underlying representation?

Abstract Data Types (2)

• **float** is an ADT, defined in terms of a set of operations (+, -, ++, etc.) not its internal representation.

  float x, y; // x and y are instances of type float

• Objects are instances of programmer defined ADT’s.

Abstract Data Types (3)

Example: Stack ADT defined in terms of operations push, pop, etc.

```java
Stack s1 = new Stack();
Stack s2 = new Stack();
s1.push(5);
s2.push(6);
s1.push(7);
int i = s1.pop(); // 7
int j = s1.pop(); // 5
int k = s2.pop(); // 6
```
Classes

• Most object-oriented languages use classes to implement ADT’s.

```java
class Stack {
    private int[] data = new int[1000];
    private int top = 0;
    public void push(int n) {
        data[top++] = n;
    }
    public int pop() {
        return data[--top];
    }
}
```

Preventing the Y2K problem?

```java
interface Date {
    // Methods for comparing dates
    public boolean before(Date d);
    public boolean after(Date d);
    public boolean equals(Date d);
    // Formatting and parsing
    public String toString();
    public void parse(String s);
    // Month, day, year manipulation
    public int getMonth();
    public void setMonth(int mon);
    public int getDay();
    public void setDay(int day);
    public int getYear();
    public void setYear(int year);
}
```

Y2K Problem (2)

```java
// Check if expiration date has passed
Date today;
TextField month, year;
Boolean expired;

// Code to initialize variables, set up GUI, etc...
... 
int mon = today.getMonth();
int yr = today.getYear();
int mon2 = Integer.parseInt(month.getText()).intValue();
int yr2 = Integer.parseInt(year.getText()).intValue();
if (yr2 + 1900 == yr)
    expired = (mon2 > mon);
else
    expired = (yr > yr2 + 1900);
```
Y2K Problem (3)

interface Date {
    // Precision constants
    public final int YEAR = 0;
    public final int MONTH = 1;
    public final int DAY = 2;

    // Methods for comparing dates
    public boolean before(Date d);
    public boolean after(Date d);
    public boolean equals(Date d);

    public boolean before(Date d, int precision);
    public boolean after(Date d, int precision);
    public boolean equals(Date d, int precision);

    // Formatting and parsing
    public String toString();
    public void parse(String s);
}

Y2K Problem (4)

// Check if expiration date has passed

Date today, expiration;
TextField month, year;
Boolean expired;

// Code to initialize variables, set up GUI, etc.

String expDate = month.getText() + "\"/\"" + year.getText();
expiration.parse(expDate);
expired = today.after(expiration, Date.MONTH);

Object-Oriented Paradigm

- Inheritance
  - Reuse mechanism by which a new class is derived from an existing class

- Overriding
  - The redefinition of an inherited method in a subclass

- Polymorphism
  - The ability of a variable to refer to objects of more than one type

- Late-binding
  - The selection of the method to invoke based on the dynamic (runtime) type of the object receiving the message
Example: Taxonomy of Animals

```java
class Animal {
    private float weight;
    boolean canFly() {
        return false;
    }
    class Mammal extends Animal {
        private Color hairColor;
    }
    class Bear extends Mammal {
    }
    class Bird extends Animal {
        boolean canFly() {
            return true;
        }
    }
    class Penguin extends Bird {
        boolean canFly() {
            return false;
        }
    }
    Animal a;
    boolean b;
    a = new Bird();
    b = a.canFly();
```

Class hierarchies

```java
Penguin p = new Penguin();
```

Implementation of OO Language

```java
class A {
    private int x;
    private int y;
    public int foo(int n) {
        return n*(x+y);
    }
}
A a1 = new A();
A a2 = a1; // OK

class B extends A {
    private int i;
    public int bar() {
        return i*(x+y); // error
    }
}
B b1 = new B();
B b2 = a1; // error
```
Example: CAD Drawing module

struct Circle {
        Point center;
        int radius;
    };

struct Rect {
        Point topLeft;
        Point bottomRight;
    };

enum { CIRCLE, RECT, ... };

struct Shape {
        int type;
        union {
            Circle c;
            Rect r;
            //...
        } shape;
    };

void drawCircle(struct Circle)
{
    //...
}

void redraw(struct Shape s[],
            int n)
{
    for (int i = 0; i < n; i++)
        switch(s[i].type) {
            case CIRCLE:
                drawCircle(s[i].shape.c);
                break;
            case RECT:
                drawRect(s[i].shape.r);
                break;
            //...
        }
}

CAD Drawing module (OO)

class Shape {
    public void draw() { }
}
class Circle extends Shape {
        Point center;
        int radius;
    public void draw() { }
    };

class Rect extends Shape {
        Point topLeft;
        Point bottomRight;
    public void draw() { }
    };

void redraw(Shape s[],
            int n)
{
    for (int i = 0; i < n; i++)
        s[i].draw();
}

Key Points

- Object-oriented programming style may be viewed as: message passing, simulation, metaphor, or delegation.
- Data abstraction is central to OOP
  - Defining and using abstract data types
- Inheritance, Polymorphism, and Late-binding promote the development of robust, reusable code.