Object-Oriented Programming

Outline

- History of programming languages
- Procedural programming vs. Object-oriented programming
- Basics concepts of OOP

Readings:
- *Core Java 2*, Section 4.1

History of programming languages

Early programming languages such as Basic, Fortran... are unstructured and allow spaghetti code. Programmers used “goto” and “gosub” to jump to anywhere in the programs.

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>k=1</td>
</tr>
<tr>
<td>20</td>
<td>gosub 100</td>
</tr>
<tr>
<td>30</td>
<td>if y &gt; 120 goto 60</td>
</tr>
<tr>
<td>40</td>
<td>k = k + 1</td>
</tr>
<tr>
<td>50</td>
<td>goto 20</td>
</tr>
<tr>
<td>60</td>
<td>print k, y</td>
</tr>
<tr>
<td>70</td>
<td>stop</td>
</tr>
<tr>
<td>100</td>
<td>y = 3<em>k^2 + 7</em>k - 3</td>
</tr>
<tr>
<td>110</td>
<td>return</td>
</tr>
</tbody>
</table>

Jump to an arbitrary position in the program

Hard to understand, error-prone, difficult to modify.

Next came the new structural programming paradigm with languages like Algol, Pascal, C...

```
int func(int j) {
    return (3*j^2 + 7*j - 3;
}
int main() {
    int k = 1
    while (func(k) < 120)
        k++;
    printf("%d\t%d\n", k, func(k));
    return(0);
}
```

Characteristics of structural / procedural programming paradigm:
- Use of looping structures: for, while, repeat, do-while
- Programs as a series of functions/procedures
- Program source code focus on representing algorithms: how to do things.
Limitation of procedural programming

- Data are separated from processes
- Passive data, active processes
  - No guarantee of data consistency and constraints
    - Hard to prevent client code from modifying library’s data
  - Difficulty in code maintenance
    - Processes are scattered, but
    - Each must understand the data structures

```c
struct Date {
    int day;
    int month;
    int year;
};
```

```c
void someProcess()
{
    ... date.day = someDay; date.month = someMonth; date.day = someYear; ...
}
```

What if someDay, someMonth, someYear make an invalid date?

Object-oriented programming

OOP helps remove those limitations

```c
class Date {
public:
    void setDate(int newDay, int newMonth, int newYear);
    int getDay() { return day; }
...
private:
    int day;
    int month;
    int year;
};
void Date::setDate(int newDay, int newMonth, int newYear)
{
    //check validity of newDay, newMonth, newYear ...
    //set new values ...
}
```

Data come together with related processing code
- Allow for guarantee of data consistency and constraining
- Easier maintenance

What is OOP?

- **Object-Oriented Programming**
  - Map your problem in the real world
  - Define “things” (objects) which can either do something or have something done to them
  - Create a “type” (class) for these objects so that you don’t have to redo all the work in defining an objects properties and behavior

```c
STUDENT
* studentNo - name - birthday + enroll() ...

CLASS
* courseID - lecturer - instructor + getStudents() + setLecturer() ...
```

The goals of object-oriented design

- **Robustness**: software is capable of handling unexpected inputs that are not explicitly defined for its application.
  - Nuclear plant control software
  - Airplane control software
- **Adaptability**: software that can evolve over time in response to changing conditions in its environment.
  - Web browsers and Internet search engines typically involve large programs that are used for many years.
- **Reusability**: the same code should be usable as a component of different systems in various applications.
  - Save time and money
Important OO concepts

- Abstraction
- Objects & Class
  - Object state and behavior
  - Object identity
  - Messages
- Encapsulation
  - Information/implementation hiding
- Inheritance
- Polymorphism

Abstraction

Abstraction: to distill a complicated system down to its most fundamental parts and describe these parts in a simple, precise language.

- naming the parts
- explaining their functionality

Examples:
- Design of data $\Rightarrow$ abstract data types (ADT)

Abstraction

Five basic characteristics of Smalltalk, the first successful object-oriented language

- Everything is an object.
- A program is a bunch of objects telling each other what to do by sending messages.
- Each object has its own memory made up of other objects.
- Every object has a type.
- All objects of a particular type can receive the same messages.

Objects

- Three components:
  - **State:**
    - Given by object’s internal data
  - **Behaviour:**
    - Produced by object’s methods
  - **Identity:**
    - Objects in any of its states are unique entities (with unique addresses in memory)
    - Different objects could have same values for data (same state)
    - An object is manipulated via its handle
      - In C++: pointers and references
      - In Java: object references

```cpp
class Date
{
public:
  void setDate(…);
  int getDay();
  ...
private:
  int day;
  int month;
  int year;
};
Date today = new Date();
today.setDate(2, 2, 2010);
```
Messages

- A means for object A to request object B to perform one method of B’s.
- A message consists of:
  - Handle of the destination object – host (today)
  - Name of the method to perform (setDate)
  - Other necessary information – arguments (2, 2, 2010)
- In fact, a message is a function call with the host object as the implicit argument (method invocation)
- However, the concept of messages has great significance to OOP
  - Data become active
  - Contrast to the old view of programming that
    - All actions are centrally controlled
    - Data are passive, procedures manipulate data

Object classes

- Classes
  - are the templates to create objects (instantiate).
  - Each object has the same structure and behaviour as the class from which it was created
- “Data type – Variable” relation
  - Classes are what we design and code
  - Objects are what are created (from a class) at run-time

Encapsulation / Information hiding

- Encapsulation: to group related things together, so as to use one name to refer to the whole group.
  - Functions/procedures encapsulate instructions
  - Objects encapsulate data and related procedures
- Information hiding: encapsulate to hide internal implementation details from outsiders
  - Outsiders see only interfaces
  - Programmers have the freedom in implementing the details of a system.
  - The only constraint on the programmer is to maintain the interface
  - public, private, and protected

Inheritance

- “is-a” relations
- The general classes can be specialized to more specific classes
- Reuse of interfaces & implementation
- Mechanism to allow derived classes to possess attributes and operations of base class, as if they were defined at the derived class
- We can design generic services before specialising them
Polymorphism

- Polymorphism:
  - Ability to assume different forms or shapes.
  - To exist in more than one form

- Object polymorphism:
  - Objects of different derived classes can be treated as if they are of the same class – their common base class
  - Objects of different classes understand the same message in different ways
    - example: on receiving message `draw()`, `Rectangle` and `Triangle` objects perform different methods

OOP languages

- Some OOP features can be implemented in C or other procedural programming languages, but not enforced by these languages
- OOP languages: OOP concepts are embedded in and enforced by the languages.
- OOP languages vary in degrees of object-oriented
  - Pure: Smalltalk, Eiffel, Ruby, JADE..
  - Original OO plus some procedural features: Python, Java (very high), C++ (mixed), C#..
  - OO features as extension: VB.NET, Fortran 2003, PHP, Perl..

OO in programming languages

Note: OOP does not give good design by default.
- Example: two pieces of code below are in effect the same. The OOP piece is even worse.

```cpp
struct Date {
    int day;
    int month;
    int year;
};

class Date {
    public:
    int getDay { return day; }
    ...;
    int setDay(int newDay) { day = newDay; }
    ...;
    private:
    int day;
    int month;
    int year;
};
```