Object-Oriented Programming in C++

Feza BUZLUCA
Istanbul Technical University
Computer Engineering Department
http://faculty.itu.edu.tr/buzluca
http://www.buzluca.info

Overview

- Introduction
- Why Object Technology?
- Quality Attributes of Software
- Software Development Process
- Unified Software Development Process (UP)
- What Is Programming?
- Learning a Programming Language
- Why C++?
- An Obsolete Technique: Procedural Programming
- The Object-Oriented Approach
- Encapsulation and Data Hiding
- Object Example: A Point In a Graphics Program
- Model of an Object and Structure of an Object-Oriented Program
- Benefits of OOP
- Conclusions
- Analogy: Learning to Play Chess - Learning to Design Software

Objectives

- To introduce fundamentals of object-oriented programming and generic programming
- To show how to use these programming schemes with C++ to build “good” programs

Need for a Good Programming Method

- Problems:
  - Costs of software projects are going up, and hardware costs are going down
  - Software development time is getting longer, and maintenance costs are getting higher
  - Software errors are getting more frequent as hardware errors become almost nonexistent
  - Too many projects have serious failures (budget, time, errors)

Cost of Software Bugs

"The National Institute of Standards and Technology (NIST) recently reported that software bugs cost American companies approximately $60 billion in 2001, while Bill Guttmann of Carnegie-Mellon University's Sustainable Computing Consortium pegs the real cost closer to three or four times that."

> - Financial Times (London) article titled "Battling the bugs: Computer software problems are costing billions and threatening to hold back hardware developments," p.12, August 27, 2002 (http://technews.acm.org/articles/2002-4/0828w.html#item8).

Why Object Technology?

Expectations are

- Reducing the effort, complexity, and **cost of development**
- Reducing the **cost of maintenance** (finding bugs, correcting them, improving the system)
- Reducing the time to **adapt an existing system** (quicker reaction to changes in the business environment
  - Flexibility, reusability
- Increasing the **reliability** of the system (fewer failures)

Object-oriented programming enables programmers to build high-quality programs.

Quality Attributes of Software

A program must
- perform its task correctly
- be useful and usable
- run as fast as necessary (real-time constraints)
- not waste system resources (processor time, memory, disk capacity, network capacity)
- be reliable
- be easy to update
- have sufficient documentation (user's manual)

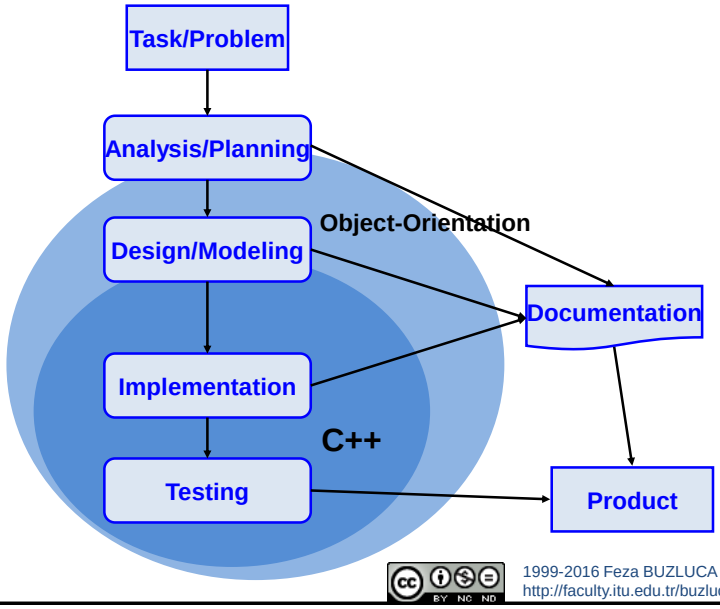
User

- Source code must be readable and understandable
- Program must be easy to maintain and update (change) according to new requirements
- An error should not affect other parts of the program (locality of errors)
- Modules should be reusable in further projects
- Software project must meet the deadline
- Software must have sufficient documentation (about development)

Software developer

Must always keep these these quality attributes in mind while designing and coding a program!

Software Development Process



Basic Steps of Software Development Process

Roles:

- Analysis: gaining clear understanding of the problem and understanding requirements
  - Requirements may change during (or after) development of system!

Analyst

- Design: identifying concepts (entities) and their relations in a solution
  - Here, our design style is object-oriented. So, entities are objects.
  - This stage has a strong effect on the quality of the software. So, before coding, created model must be verified.

Software architect, designer

- Coding: expressing the solution (model) in a program
  - Coding is related to the programming language
  - In this course, we will use C++

Developer

- Documentation: clear explanation of each phase of software project
  - A user's manual should also be written

Training

- Testing: examination of behavior of each object and of whole program for possible inputs

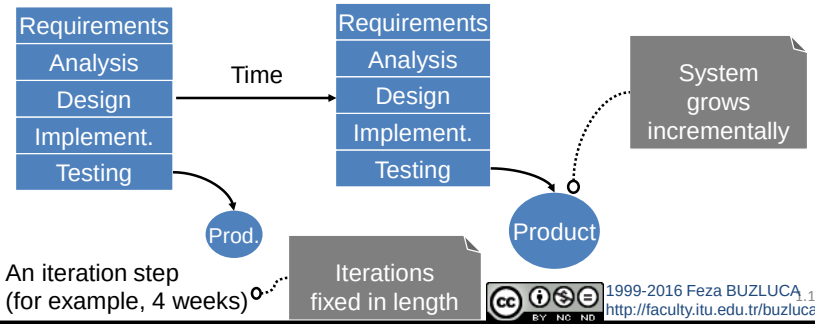
Quality Assurance, Tester

Software Development (SD) Process

- Describes an approach to building, deploying, and possibly maintaining software
- Details covered in "Software Engineering" course

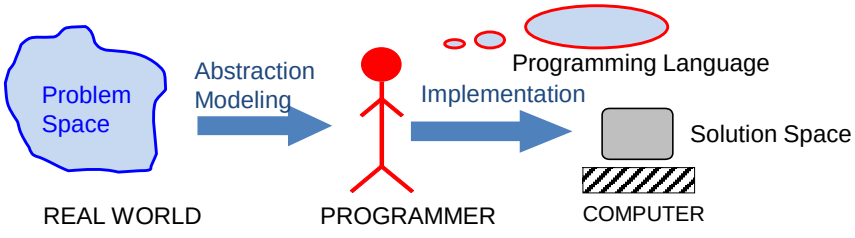
Unified (SD) Process (UP)

- Is a popular iterative software development process for building object-oriented systems
- Promotes several best practices
  - Iterative: Development is organized into a series of short, fixed-length (for example, three-week) mini-projects called iterations; the outcome of each is a tested, integrated, and executable partial system. Each iteration includes its own requirements analysis, design, implementation, and testing activities.
  - Incremental, evolutionary
  - Risk-driven



What Is Programming?

- Like any human language, a programming language provides a way to express concepts
- Program development involves creating models of real-world situations and building computer programs based on these models
- Computer programs describe the method of implementing the model
- Computer programs may contain computer world representations of the things that constitute the solutions of real-world problems



- If successful, the object-oriented way will be significantly easier, more flexible, and more efficient than alternatives as problems grow larger and become more complex

Learning a Programming Language

- Knowledge of grammar rules (syntax) of a programming language not enough to write “good” programs
- Focusing on concepts and not getting lost in language-specific details essential
- Design techniques far more important than an understanding of details
- Understanding of details comes with time and practice
- Before the rules of the programming language, the programming technique must be understood

Which Compiler?

- Be aware of programming standards and use compilers which support the most current one
- For this course, use compilers which support latest C++ standard, ISO/IEC 14882 (currently: 2014 “C++14”)
  - This standard is available for download on campus from the Web site of British Standards Online: http://bsol.bsigroup.com/

Why C++?

The main objective of this course is not to teach a programming language; however, examples are given in C++ due to its

- support for object-orientation and generic programming
- high performance (especially, speed) of C++ programs
- usage by hundreds of thousands of programmers in every application domain
  - Hundreds of libraries
  - Hundreds of textbooks, several technical journals, many conferences
- ease with which C++ programmers can adapt to other object-oriented programming languages such as Java or C#.

Application Domain of C++

- Systems programming
  - Operating systems, device drivers
  - Here, direct manipulation of hardware under real-time constraints is important
- Banking, trading, insurance
  - Maintainability, ease of extension, reliability
- Graphics and user interface programs
- Computer communications programs

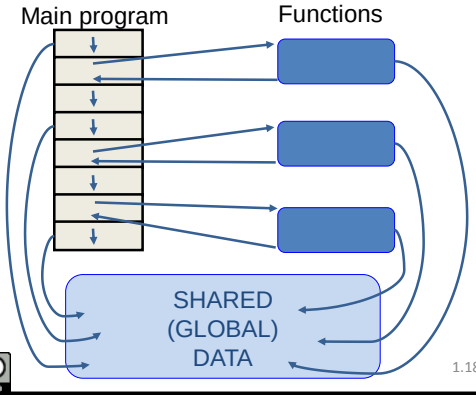
Examples That Use C++

Written in part or in their entirety with C++:

- Apple's Mac OS X
- Adobe Illustrator
- Facebook
- Google's Chrome browser
- Microsoft Windows operating systems
- Internet Explorer
- Firefox
- MySQL

An Obsolete Technique: Procedural Programming

- Pascal, C, BASIC, Fortran, and similar traditional programming languages are procedural languages
- Each statement in the language tells the computer to do something
- In a procedural language, emphasis is on **doing things (functions)**
- A program is divided into **functions**, and (ideally, at least) each function has a clearly-defined purpose and a clearly-defined interface to the other functions in the program



Problems with Procedural Programming

- Data is undervalued
  - Data is, after all, the reason for a program's existence programming
  - For example, the important parts of a program about a school are not functions that display the data or functions that check for correct input; they are student and teacher data
- Procedural programs (functions and data structures) do not model the real world very well
  - The real world does not consist of functions
- Functions that have no business changing global data could actually corrupt the global data
- To add new data items, all functions that access data must be modified so they can also access these new items
- Creating new data types is difficult

It is also possible to write good programs using procedural programming (e.g., C programs), but object-oriented programming offers programmers many advantages enabling them to write high-quality programs

The Object-Oriented Approach

- Fundamental idea behind object-oriented programming: The real world consists of objects
- Computer programs may contain computer world representations of the things (objects) that constitute the solutions of real world problems
- Thinking in terms of objects rather than functions has a helpful effect on design process of programs
- This results from the close match between objects in the programming sense and objects in the real world (low representational gap)

Example: A University

In a university system, there are many functions and a high level of complexity:

- Students have ID numbers. They attend classes and get grades. The system calculates their GPAs.
- Instructors teach courses, carry out research projects, and have administrative duties. Their salaries are calculated each month.
- Courses are assigned to specific time slots and classrooms. Courses have a plan and a list of enrolled students.
- Looking at every element at once, a university system becomes very complex.

The Object-Oriented Approach (cont.): Encapsulation and Data Hiding

- If you wrap what you see in the problem into objects, the system is easier to understand and handle. There are students, instructors, and courses.
- Internal mechanisms and various parts that work together are wrapped up into an *object*.
- To solve a programming problem in an object-oriented language, the programmer asks how it will be divided into objects.

The Object-Oriented Approach (cont.): Encapsulation and Data Hiding

Real-world objects have two parts:

1. Attributes (*property* or *state*: characteristics that can change)
2. Behavior (or *abilities*: things they can do or *responsibilities*).

The Object-Oriented Approach (cont.): Encapsulation and Data Hiding

What kinds of things become objects in object-oriented programs?

- Human entities: Employees, customers, salespeople, workers, managers
- Graphics program: Point, line, square, circle, ...
- Mathematics: Complex numbers, matrices
- Computer user environment: Windows, menus, buttons
- Data-storage constructs: Customized arrays, stacks, linked lists

The Object-Oriented Approach (cont.): Encapsulation and Data Hiding

- Encapsulation: To create software models of real-world objects both *data* and the *functions* that operate on that data are combined into a single program entity.
- Data represent the attributes (state), and functions represent the behavior of an object.
- Data and its functions are said to be *encapsulated* into a single entity (class).
- An object's functions, called *member* functions in C++, typically provide the only way to access its data.

The Object-Oriented Approach (cont.): Encapsulation and Data Hiding

- The data is usually hidden, so it is safe from accidental alteration.
- If you want to modify the data in an object, you know exactly what functions interact with it: the member functions in the object. No other functions can access the data.
- This simplifies writing, debugging, and maintaining the program.
- Encapsulation and data hiding are key terms in the description of object-oriented languages.
- Other important concepts of OOP are inheritance and polymorphism, which will be explained in subsequent lectures.

Object Example: A Point in a Graphics Program

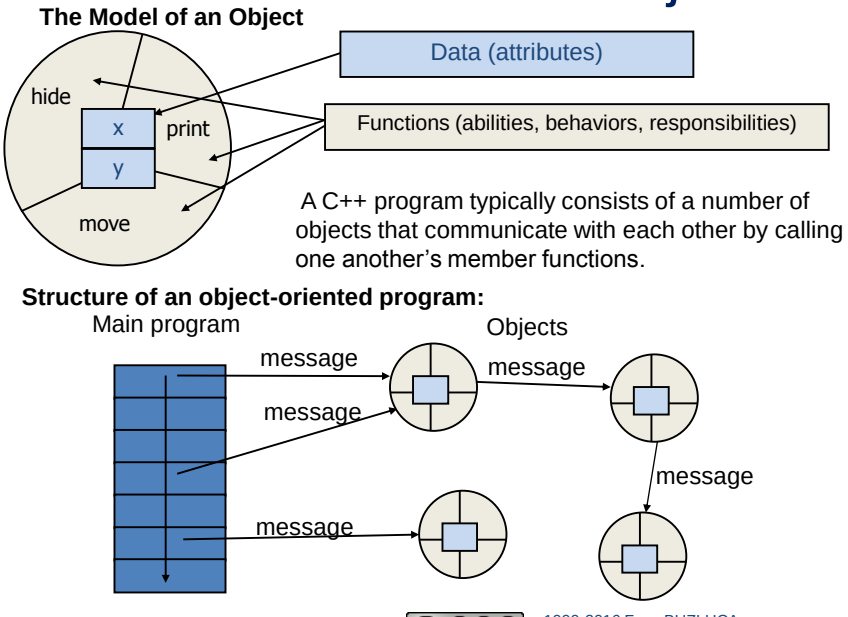
- A point on a plane has two attributes: x- and y-coordinates.
- Abilities (behavior, responsibilities) of a point are moving on the plane, appearing on the screen, and disappearing.
- We can create a model for 2-dimensional points with the following parts:
  - Two integer variables (x, y) to represent x- and y-coordinates
  - A function to move the point: move
  - A function to print the point on the screen: print
  - A function to hide the point: hide

Object Example: A Point in a Graphics Program

- Once the model has been built and tested, it is possible to create many objects of this model in the main program.

```
Point point1, point2, point3;
    :
point1.move(50,30);
point1.print();
```

The Model of an Object



Conclusion 1 (Good News)

- The object-oriented approach provides tools for the programmer to represent elements in the problem space.
- We refer to elements in the problem space (real world) and their representations in the solution space (program) as "objects."
- OOP allows us to describe the problem in terms of the problem, rather than in terms of the computer where the solution will run.
- So, when we read the code describing the solution, we are reading words that also express the problem.

Benefits of OOP (if applied properly)

- Understandability: It easy to understand a good program. As a result, it is easy to analyze the program for causes of failures and modify it if necessary.
- Low probability of errors
- Maintenance: It easy to add new modules (parts of the software system) or modify existing modules.
- Reusability: Existing modules can be used in new projects.
- Teamwork: Modules can be written by different members of the team and can be integrated easily.

Conclusion 2 (Bad News)

- Programming is fun but is related mostly to the implementation phase of software development.
- Development of high-quality software is a bigger job and besides programming skills other capabilities are also necessary.
- In this course, we will cover OO basics: encapsulation, data hiding, inheritance, polymorphism.
- Although OO basics are important building blocks, a software architect must also be aware of design principles and software design patterns, which help us in developing high-quality software.
  - See chess vs. software analogy in the next slides.
- Design principles and patterns are covered in another course:
  - Object Oriented Modeling and Design (http://www.ninova.itu.edu.tr/tr/dersler/bilgisayar-bilisim-fakultesi/2097/blg-468e/)
- The Unified Modeling Language (UML) is a useful tool to express the model.

Analogy: Learning to Play Chess-Learning to Design Software

Chess:

1. Basics: Rules and physical requirements of the game, the names of all the pieces, the way that pieces move and capture.
   - At this point, people can play chess, although they will probably not be very good players.
2. Principles: The value of protecting the pieces, the relative values of those pieces, and the strategic value of the center squares.
   - At this point, people can play a good game of chess.
3. Studying the games of other masters (Patterns): Buried in those games are patterns that must be understood, memorized, and applied repeatedly until they become second nature.
   - At this point, people can become masters of chess.

This chess analogy has been borrowed from Douglas C. Schmidt
http://www.cs.wustl.edu/~schmidt/

Analogy: Learning to Play Chess-Learning to Design Software

Software:

1. Basics: The rules of the languages, data structures, algorithms.
   - At this point, one can write programs, albeit not very good ones.
2. Principles: Object-oriented programming.
   - Importance of abstraction, information hiding, cohesion, dependency management, etc.
3. Studying the games of other masters (Patterns): Deep within those designs are patterns that can be used in other designs.
   - Those patterns must be understood, memorized, and applied repeatedly until they become second nature.

This chess analogy has been borrowed from Douglas C. Schmidt
http://www.cs.wustl.edu/~schmidt/