OBJECT-ORIENTED PROGRAMMING IN C++

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

INTRODUCTION

Main Objectives of the Course :

- To introduce **Object-Oriented Programming** and **Generic Programming**
- To show how to use these programming schemes with the C++ programming language to build "**good**" (high-quality) programs.

Need for high-quality design and good programming methods:

Problems:

- Software project costs (especially maintenance costs) are getting higher, while hardware costs are going down.
- Software errors are getting more frequent as hardware errors become almost non-existent.

The software crisis report by Standish Group in 2015:

- Only 29% of all projects succeeding by delivered on time, on budget, with required features and functions (with a satisfactory result).
- 52% of software projects were late, over budget, and/or with less than the required features and functions.
- 19% of projects were failed and were cancelled prior to completion, or delivered and never used.

Goal of a software development Project:

The ability to deliver a software system

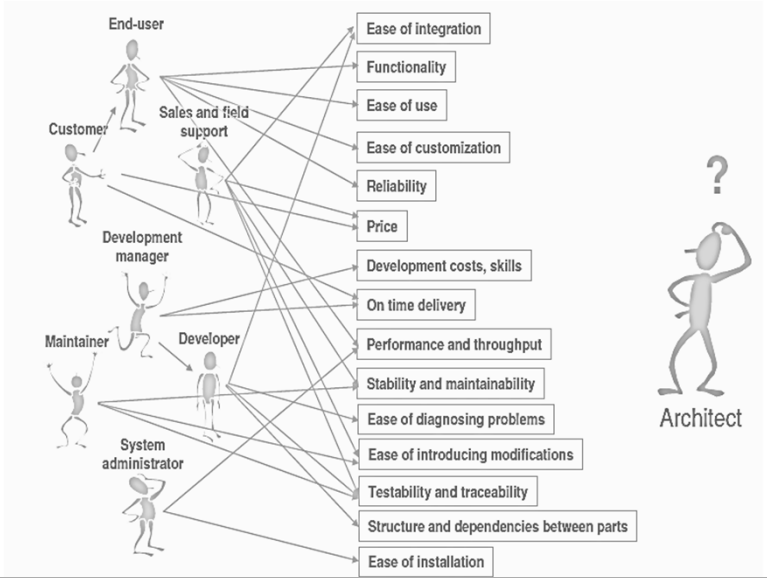
1. that meets **quality** needs of different stakeholders (user, developer, customer …)
   - Functionality
   - Performance (speed, accuracy, etc.)
   - Efficiency (processor, memory, network etc.)
   - Reliability (error free)
   - Security (access control)
   - Maintainability (modify, extend, reuse)
   - …

   } Some of the software quality attributes
2. on time,
3. within budget.

Just writing a code that runs somehow is not sufficient!

You should consider the quality needs and expectations of the stakeholders of the system.

Expectations of different stakeholders (Quality needs):



Source: D. Falessi, G. Cantone, R. Kazman, and P. Kruchten, "Decision-making techniques for software architecture design," *ACM Computing Surveys*, vol. 43, pp. 1-28, Oct. 2011.

Quality characteristics of a software system

ISO (the International Organization for Standardization) and **IEC** (the International Electrotechnical Commission) prepared standards for quality models.

You may find definitions of the quality attributes of a software system in the following standard.

ISO/IEC 25010: *Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*

This standard includes 2 quality models.

A) Quality in use model:

This is the external quality of the system; impact on stakeholders (customer, direct and indirect users etc.) in specific contexts of use.

B) Product Quality:

These characteristics relate to the software development team.

Details of the quality models are out of scope of this course.

They will be covered in "Object-Oriented Modeling and Design" (8th semester), and "Software Design Quality " (graduate) courses.

In this course we will give only a brief insight about the quality attributes of a software system that must be always considered during software development.

Quality Attributes of a Software (External and Internal)

- A program must do its job correctly (effectiveness). ***External***
- A program must perform as fast as necessary (Time constraints).
- It must not waste system resources (processor time, memory, disk capacity, network capacity) too much (efficiency).
- It must be reliable (trustful).
- It must be useful, usable and include sufficient documentation (easy to learn and use)
- It must be easy to update the program (flexibility). ***User***

- It must be functionally complete and correct.
- It must be efficient (time behavior, resource utilization, capacity). ***Internal***
- Source code must be readable and understandable (comments, documentation).
- It must be easy to extend and update (change) the program according to new requirements.
- It must be easy to test the program, to find and correct errors.
- Modules of the program must be reusable in further projects.

Software developer

While designing and coding a program (***and learning a programming language***), these quality attributes must be always kept in mind.

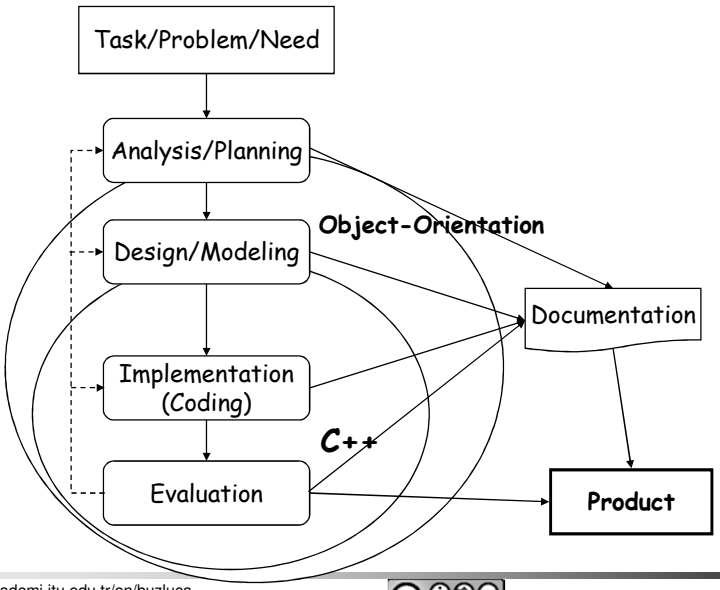
Why Object-Oriented Technology?

Expectations are,

- Reducing the effort, complexity, and **cost of development**.
- Reducing the **cost of the maintenance** (finding bugs, correcting them, improving the system).
- Reducing the **cost of extending the system** (adding new features).
- Reducing the effort to **adapt an existing system** (quicker reaction to changes in the business environment) (Flexibility).
- Reducing the effort to **use existing models in a new project (r**eusability).
- Increasing the **reliability** of the system (Fewer failures.)

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

Software Development Process



Basic steps of the software development process

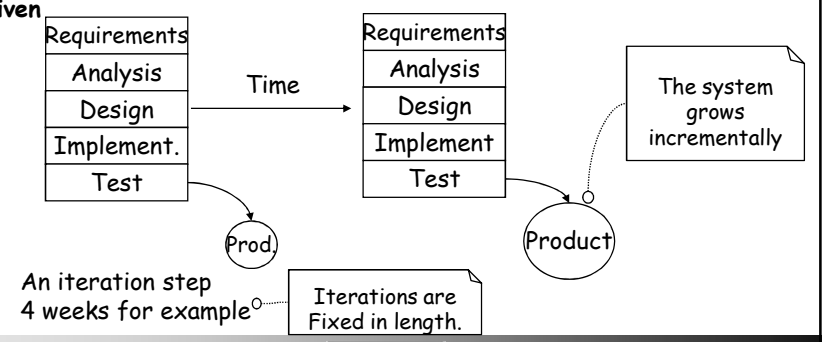
- **Analysis:** Gaining a clear understanding of the problem. (Role: Analyst)
  Understanding requirements. Understanding what the user wants. Requirements may change during (or after) development of the system!
  Understanding the system (the problem). What should the system do?
- **Design:** Identifying the concepts (entities) and their relations involved in a solution. (Role: Software architect, designer)
  Here, our design style is object-oriented. So entities are objects (classes).
  This stage has a strong effect on the quality of the software.
- **Implementation (Coding):** The solution (model) is expressed in a program. (Role: Developer)
  Coding is connected with the programming language. In this course we will use C++.
- **Documentation:** Each phase of a software project must be clearly explained.
- **Evaluation:** Testing, measurement, performance analysis, quality assessment.
  The behavior of each object and of the whole program for possible different cases must be examined. (Role: Quality assurance, Tester)

Details of the software development process are covered in the "**Software Engineering**" course.

The Unified (Software Development) Process - UP

A software development process describes an approach to building, deploying, and possibly maintaining software.
The Unified Process is a popular iterative software development process for building object-oriented systems. It 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**

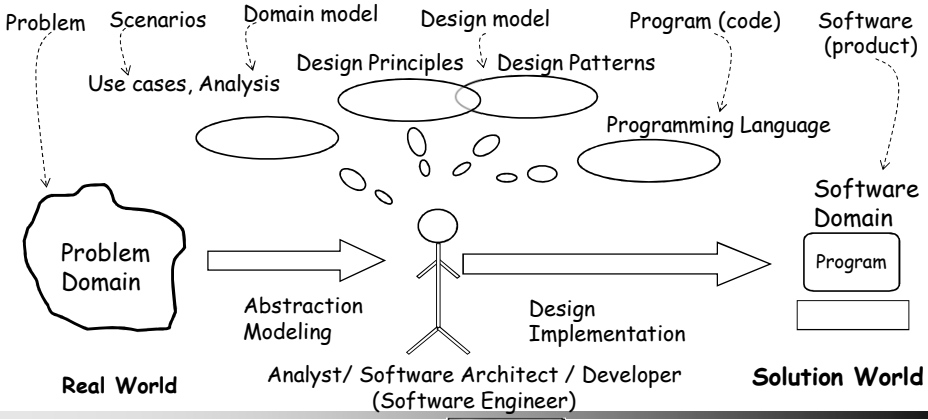


What is programming? Steps of software development

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 may contain computer world representations of **the things (objects)** that constitute the solutions of real world problems.



Learning a Programming Language

- Knowledge about grammar rules (syntax) of a programming language is not enough to write "good" programs.
- The most important thing to do when learning programming is to focus on concepts (**design and programming techniques**) and not get lost in language-technical details.
- Rather than the rules of the programming language, the programming scheme must be understood.

  Understanding of design techniques comes with time and practice.
- Always consider quality characteristics (understandability, flexibility, …).

Which Compiler?

Be aware of programming standards and use compilers that support the most current one.

The current C++ standard is ISO/IEC 14882:2017 (**C++17**).

C++20 is in progress.

You can get the standard in İTÜ campus from the web site of the British Standards Online: http://bsol.bsigroup.com/

Information about C++ standards: https://isocpp.org/std/the-standard

Why C++?

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

Properties of the C++ programming language:

- C++ supports object-orientation and generic programming.
- Performance (especially speed) of programs written with C++ is high.
- It is useful in the low-level programming environments, where direct control of hardware is necessary.

  Embedded systems and compiler are created with the help of C++.
- C++ gives the user control over memory management (also increases the responsibility of the programmer "*with authority comes responsibility*").
- C++ is used by hundreds of thousands of programmers in every application domain.
  - This use is supported by hundreds of libraries,
  - hundreds of textbooks, several technical journals, many conferences.
- C++ programmers can easily adapt to other object-oriented programming languages such as Java or C#.

Why C++? (cont'd)

Application domain of C++:

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

Examples of applications written in C++:

Apple's Mac OS X,

Adobe Systems,

Backend services of Facebook,

Google's Chrome browser,

Microsoft Windows operating systems, MS Office, Visual Studio, Internet Explorer,

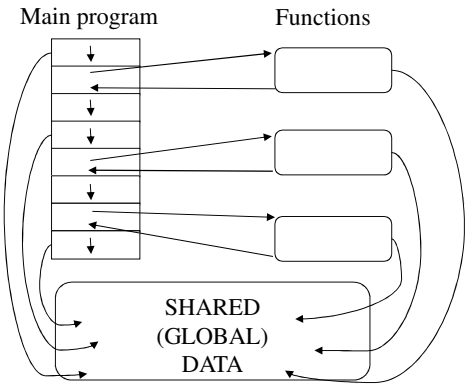
Mozilla Firefox, Thunderbird,

MySQL

are written in part or in their entirety with C++.

An Obsolete Technique: Imperative/Procedural Programming Technique

- Pascal, C, BASIC, Fortran, and similar traditional programming languages are imperative languages. That is, each statement (command) in the language tells the computer to do something.
- In a imperative/ procedural language, the emphasis is on **doing things (functions/procedures)**.
- A program is divided into **functions** (procedures) 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 the Imperative/Procedural Programming

- Procedural programs (functions and data structures) do not model the real world very well.

  The real world does not consist of only functions. The real world consists of objects.
- Data is undervalued, emphasis is on functions.
- Data is, after all, the reason for a program's existence.

  The important parts of a program about a school for example, are not functions that display the data or functions that checks for correct input; important parts are student, teacher, course data.
- Data items and related functions are scattered around the program (they are not in the same module like objects).
- Global data can be corrupted by functions that have no business changing it.
- Creating new (user-defined) data types is difficult.

It is also possible to write good programs by using procedural programming (C programs).

However, object-oriented programming offers programmers many advantages, to enable them to write high-quality programs.

The Object-Oriented Approach

The fundamental idea behind object-oriented programming is:

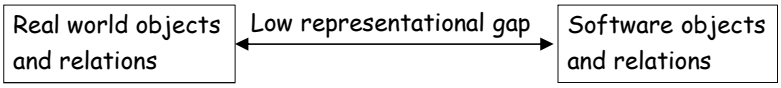
The real world (problem) consists of objects.

Computer programs may contain computer world representations of the things (objects) that constitute the solutions of real world problems.

The software system (solution) consists of objects.

Thinking in terms of objects: To solve a programming problem in an object-oriented language, the programmer asks **how it will be divided into objects**.

Low representational gap: Close match between objects in the programming sense and objects in the real world increases the quality of the design.



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

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

The Object-Oriented Approach (cont'd)

Example:

If you take a look at a university system there are many functions and a big amount of complexity.

- Students have IDs, they attend to courses, they take grades, their GPAs are calculated.
- Instructors give courses, they perform some industrial and scientific projects, they have administrative duties, their salaries are calculated each month.
- Courses are given in specific time slots in a classroom. They have a plan, they have a list of students.

Considered this way, looking at every element at once, and focusing on functions, a university system becomes very complex.

Object-oriented modeling:

If you wrap what you see in the problem up into **objects**, the system is easier to understand and handle.

There are students, instructors, courses, class rooms and relations between them.

Students take courses; courses are give in class rooms…

Internal mechanisms and various parts that work together are wrapped up into an *object (class).*

The Object-Oriented Approach (cont'd)

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*).

Examples:

- Object: Student

Attributes: ID, Name, Birthdate, List of taken courses, …

Behavior (responsibilities): Calculating her GPA, listing the course names

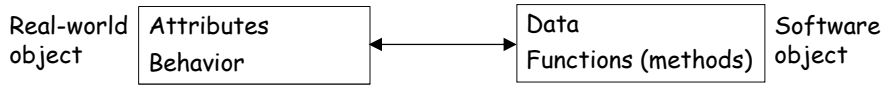
- Object: Class room

Attributes: Capacity, time table

Behavior (responsibilities): Entering date to the time table, showing the time table

Software objects (classes) have also two parts like real-world objects:

1. ***Data*** *represent attributes*,
2. ***Functions (methods)*** represent behavior.



The Object-Oriented Approach (cont'd):

Key Terms: Encapsulation - 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 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.

The other important concepts of the OOP are ***inheritance*** and ***polymorphism***, which are explained in subsequent chapters.

Example of an Object: A Point in a graphics program

A Point on a plane has two attributes; x-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** .

(The Unified Modeling Language (**UML**) is a useful tool to express the model.)

Model (class) of a point:

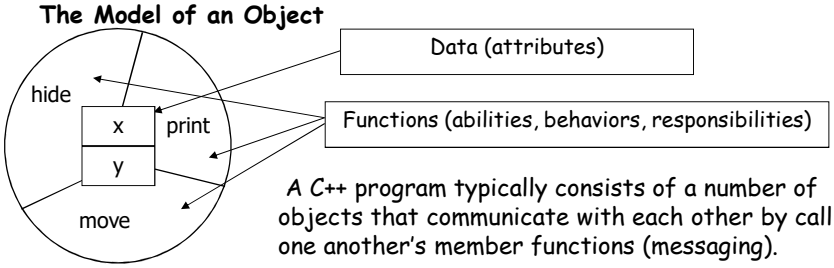
Point
- int x, y;
+ void move(int, int); + void print() + void hide()

Once the model (class) of the point has been built and tested, it is possible to create and activate many point objects from this model.

In the example on the right point1, point2, point3 are three different objects of the same class (model) Point.

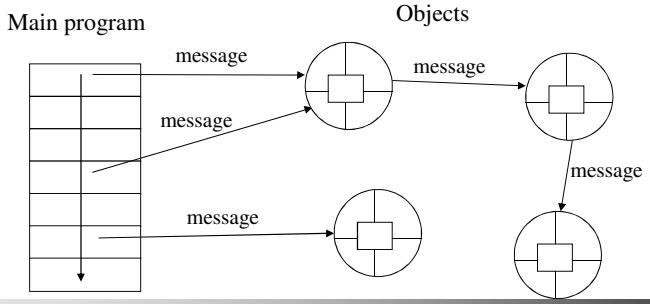
```
Point point1, point2, point3;
   :
point1.move(50,30);
point1.print();
point2.move(0,100);
```


The Model of an Object



A C++ program typically consists of a number of objects that communicate with each other by calling one another's member functions (messaging).

Structure of an object-oriented program in C++:



Conclusion 1 (Good news)

- The object-oriented approach provides tools for the programmer to represent elements in the problem space (*Low representational gap*).
- We refer to the elements in the problem space (*real world*) and their representations in the solution space (*program*) as "objects."
- OOP allows you to describe the problem in terms of the problem, rather than in terms of the computer where the solution will run.
- So when you read the code describing the solution, you're reading words that also express the problem.
- Some benefits of the OOP if the techniques are applied properly:
  - Understandability: It is easy to understand a good program. As a consequence it is easy analyze the program in case of failures and modify it if necessary.
  - Low probability of errors
  - Flexibility: 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 integrates easily.

Conclusion 2 (Bad news)

- Programming is fun but it is related (only) to the implementation phase of software development.
- Development of 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 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 (8th semester).

  http://www.ninova.itu.edu.tr/tr/dersler/bilgisayar-bilisim-fakultesi/2097/blg-468e/

Analogy: Learning to play chess – Learning to design software

Chess:

1. **Learning 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. **Learning principles:** The value of protecting the pieces, the relative value of those pieces, and the strategic value of the center squares.

   At this point, people can become **good players** 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 be **master** of chess.

Learning to play chess – Learning to design software (cont'd)

Software:

1. **Learning basics:** The rules of the languages, data structures, algorithms.

   At this point, one can write programs, albeit not very good ones.

2. **Learning principles:** Object oriented programming.

   Importance of abstraction, information hiding, cohesion, dependency management, etc.

3. **Studying the designs 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 barrowed from Douglas C. Schmidt

https://github.com/douglascraigschmidt