# OBJECT-ORIENTED PROGRAMMING IN C++

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







## INTRODUCTION

**Objectives:**

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

**Need for a good programming method:**

**Problems:**

- Software projects costs 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 non-existent.
- Too many projects have serious failures (Budget, time, errors).

**Financial Times (27/08/2002) P. 12;**
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**.


## Why Object 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 time to **adapt an existing system** (quicker reaction to changes in the business environment). Flexibility, reusability.
- Increasing the **reliability** of the system (Less failures.)

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







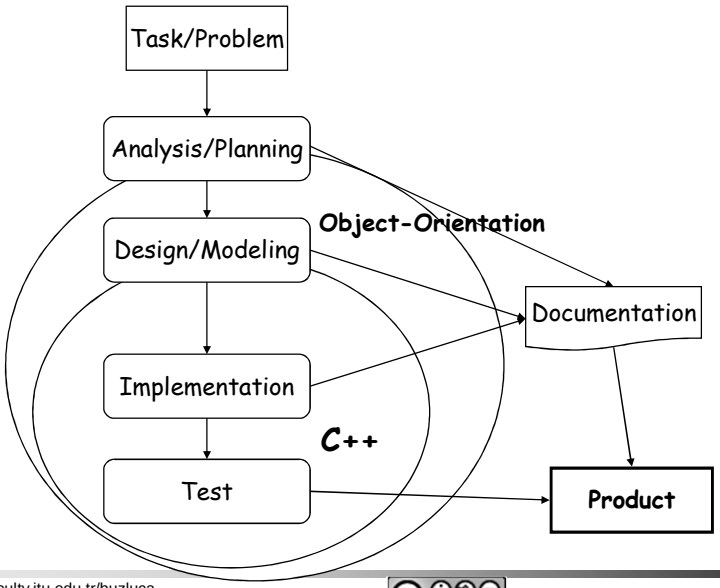
## Quality Attributes of a Software

- A program must do its job correctly. It must be useful and usable.
- A program must perform as fast as necessary (Real-time constraints).
- A program must not waste system resources (processor time, memory, disk capacity, network capacity) too much.
- It must be reliable.
- It must be easy to update the program.
- A good software must have sufficient documentation (users manual). **user**

- Source code must be readable and understandable.
- It must be easy to maintain and update (change) the program according to new requirements.
- An error may not affect other parts of a program (Locality of errors).
- Modules of the program must be reusable in further projects.
- A software project must be finished before its deadline.
- A good software must have sufficient documentation (about development). **Software developer**

While designing and coding a program, these quality attributes must be kept always in mind.

## Software Development Process








## Basic steps of the software development process

**Analysis:** Gaining a clear understanding of the problem.
Understanding requirements. They may change during (or after) development of the system! (Role: Analyst)

**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.

This stage has a strong effect on the quality of the software. Therefore, before the coding, verification of the created model must be done.

**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. A users manual should be also written. (Role: Training)

**Test:** At the end, the behavior of each object and of the whole program for possible inputs must be examined. (Role: Quality assurance, Tester)

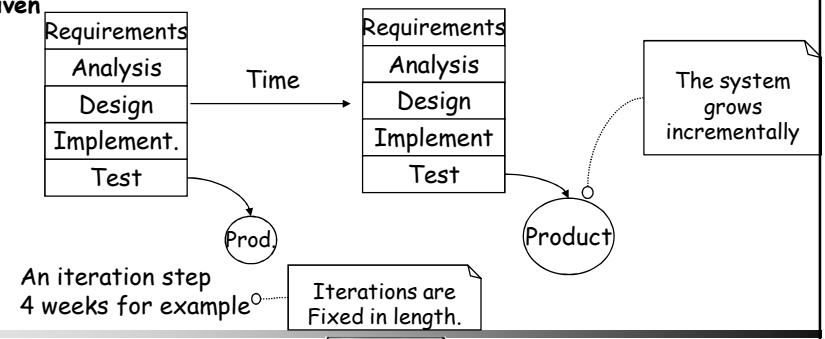
Details of the software development process are covered in the "**Software Engineering**" course

**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**









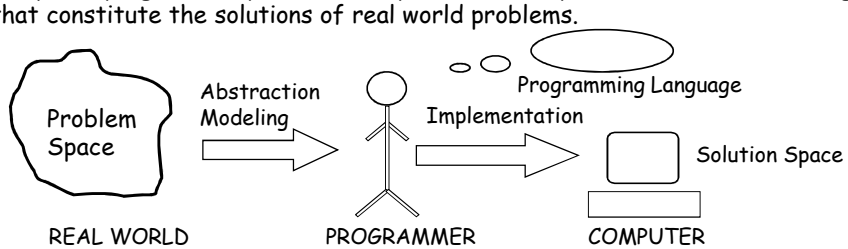

**What is programming?**

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 efficient than the alternatives as problems grow larger and more complex.

**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 and not get lost in language-technical details.

Design techniques are far more important than an understanding of details; that understanding comes with time and practice.

Before the rules of the programming language, the programming scheme must be understood.

**Which Compiler?**

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

For this course, use the compilers, which support the latest C++ standard, ISO/IEC 14882 (currently :2014 "C++14").

You can get the standard in İTÜ campus from the web site of the 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++ .

- C++ supports object-orientation and generic programming.
- Performance (especially speed ) of programs written with C++ is high.
- 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#.
- Application domain:
  - 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

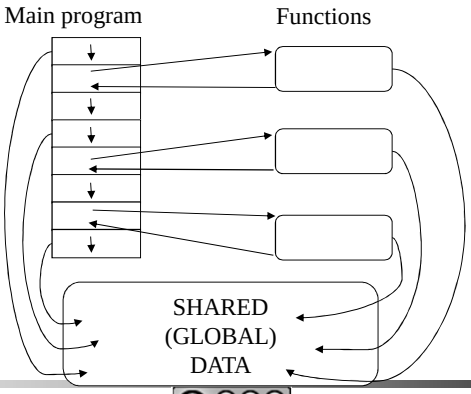
Apple's Mac OS X, Adobe Illustrator, Facebook, Google's Chrome browser, Microsoft Windows operating systems, Internet Explorer, Firefox, and MySQL are written in part or in their entirety with C++.

**An Obsolete Technique: Procedural Programming Technique**

•Pascal, C, BASIC, Fortran, and similar traditional programming languages are procedural languages. That is, each statement in the language tells the computer to do something.

• In a procedural language, the 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. 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; they are student, teacher data.
- Procedural programs (functions and data structures) don't model the real world very well. The real world does not consist of functions.
- Global data can be corrupted by functions that have no business changing it.
- To add new data items, all the functions that access the data must be modified so that they can also access these new items.
- Creating new data types is difficult.

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

**For example**, if you take a look at a university system there are many functions and a big amount of complexity.

Students have a number, 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, a university system becomes very complex.



## The Object-Oriented Approach , cont'd

If you wrap what you see in the problem up 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**.

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*).

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
• Computer user environment: Windows, menus, buttons
• Data-storage constructs: Customized arrays, stacks, linked lists

The Object-Oriented Approach (cont'd):

**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 for 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** ,

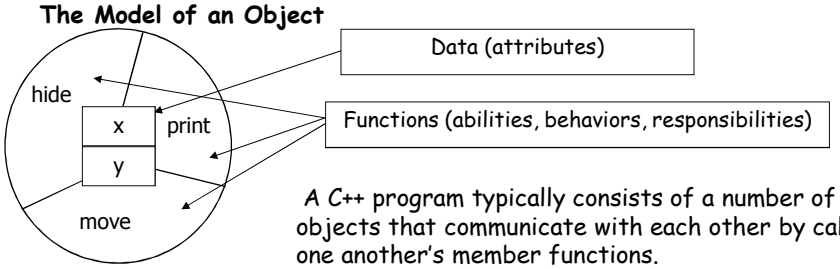
A function to print the point on the screen: **print** ,

A function to hide the point: **hide** .

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

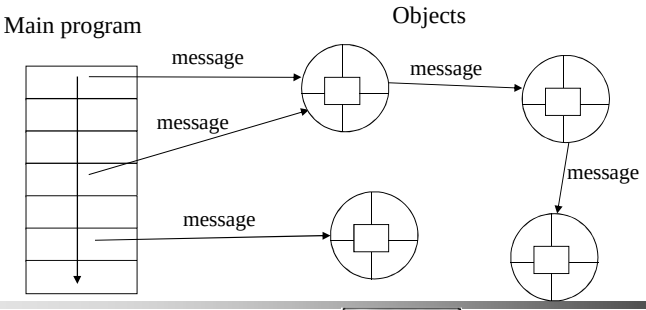
```
Point point1, point2, point3;
   :
point1.move(50,30);
point1.print();
```

**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.

**Structure of an object-oriented program:**








**Conclusion 1 (Good news)**

- The object-oriented approach provides tools for the programmer to represent elements in the problem space.
- 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.

• Benefits of the OOP if the techniques are applied properly:

- Understandability: It easy to understand a good program. As a consequence it is easy 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 integrates easily.

**Conclusion 2 (Bad news)**

- Programming is fun but it is related mostly 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**.

  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 a very good players.
2. **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 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 be master of chess.

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

**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 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*
http://www.cs.wustl.edu/~schmidt/