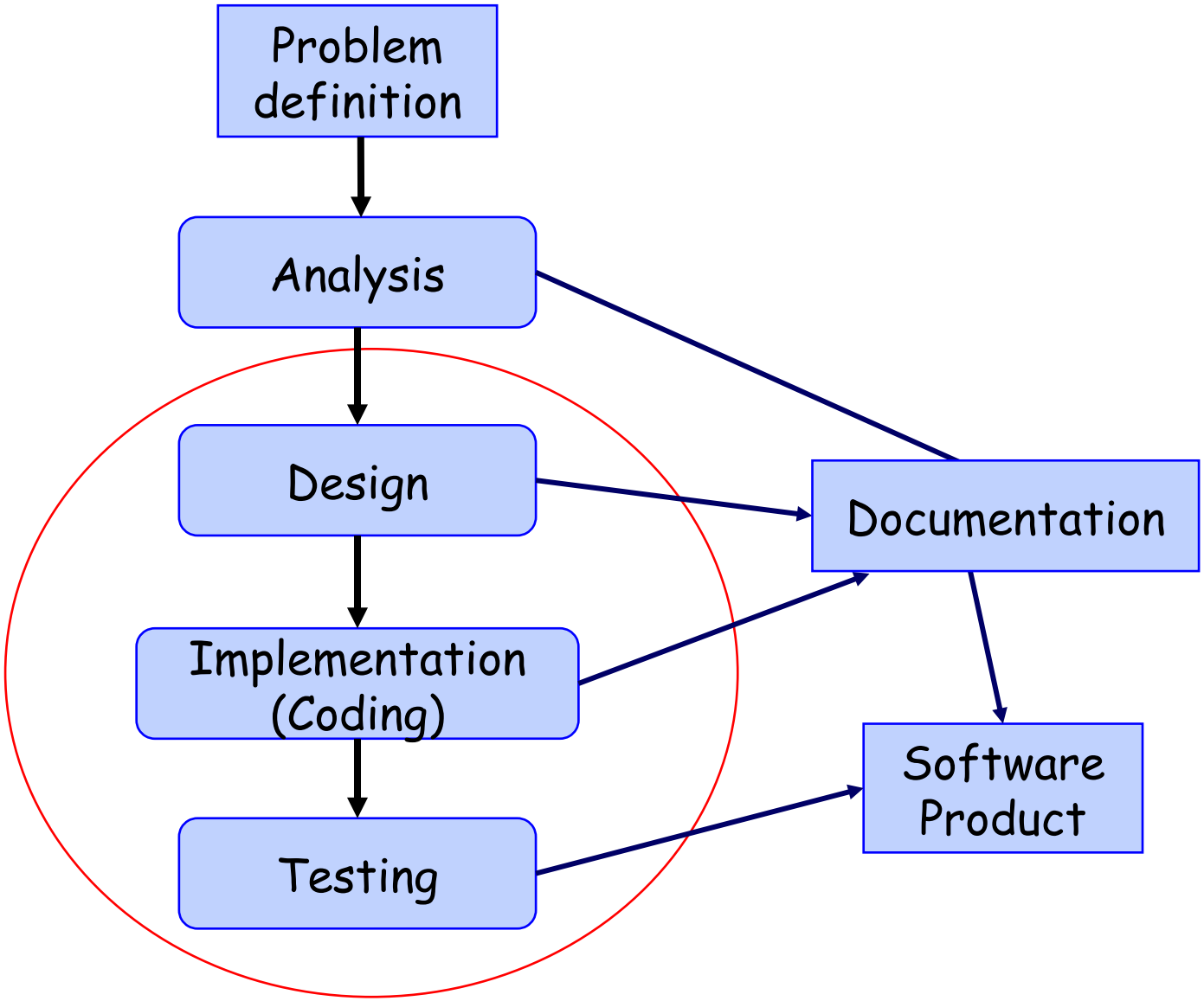
Lecture 1

Introduction

Outline

- Software Development Process
- Object Oriented Approach
- Principles of Object Oriented Programming
- Example: Graphics drawing program

Software Development Process



Software Development Process

ANALYSIS: Understanding the requirements for given problem.

DESIGN: Identifying the entities.
In object-oriented design, entities are **objects**.

CODING: Implementation in a programming language.

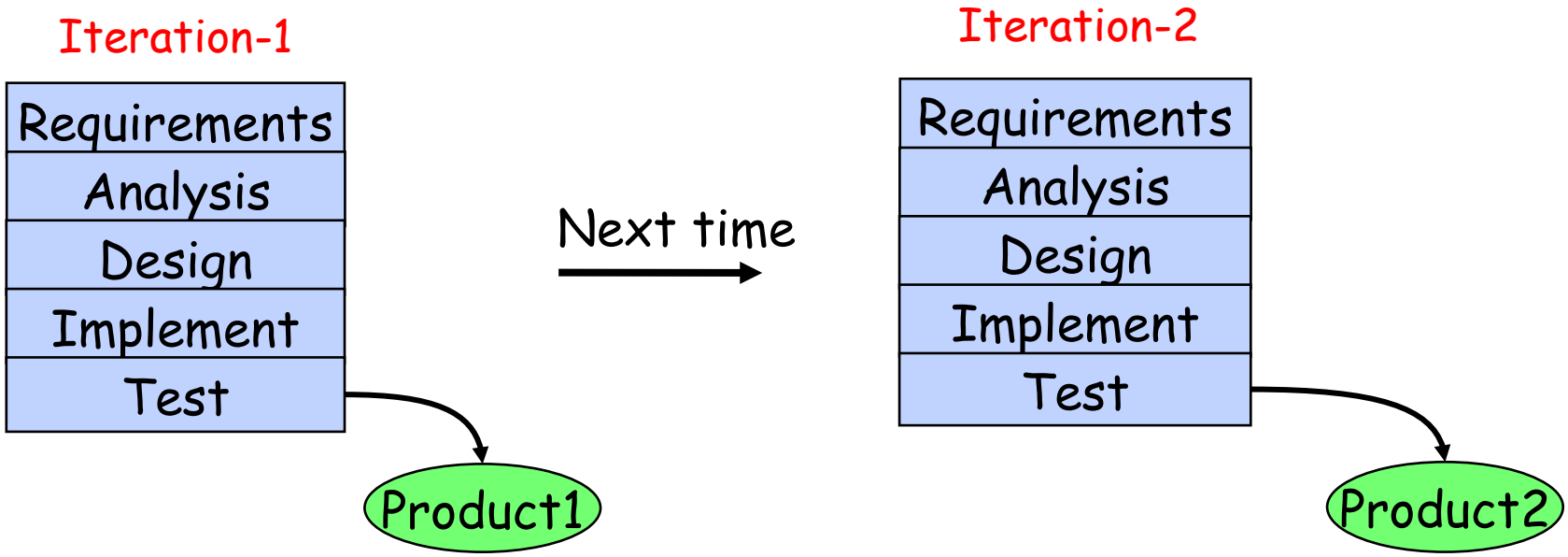
DOCUMENTATION: Writing reports for development team.
(Also a User's Manual should be written.)

TESTING: The functions of each object and the whole program must be tested for possible inputs and expected outputs.

Unified Process Method in Software Development

- A software development process describes an approach to building, deploying, and maintaining a software.
- The **Unified Process (UP)** is a popular **iterative** software development process for building object-oriented systems.
- Development is organized into a series of mini-projects called iterations.
- Each iteration includes its own analysis, design, implementation, and testing activities.

The Unified Process (UP)



An iteration has a fixed time.
(For example : 3 weeks)

User view of Program features

A program must have the following features:

- Runs **correctly**.
- Runs **reliably**.
- Performs as **fast** as necessary.
- Does not waste system **resources** too much. (Processor time, Memory, Disk).
- **Easy to up-grade** the program (re-installation).
- Have sufficient **documentation** of users manuals.

Software developer view of Program features

A program must have the following features:

- Source code must be **readable and understandable.**
- It must be **easy to maintain and update** (change) the program, according to new requirements.
- An error should not affect other parts of a program.
- Modules of program must be **reusable** in further projects.
- It must have sufficient **design documentation.**

Programming Process

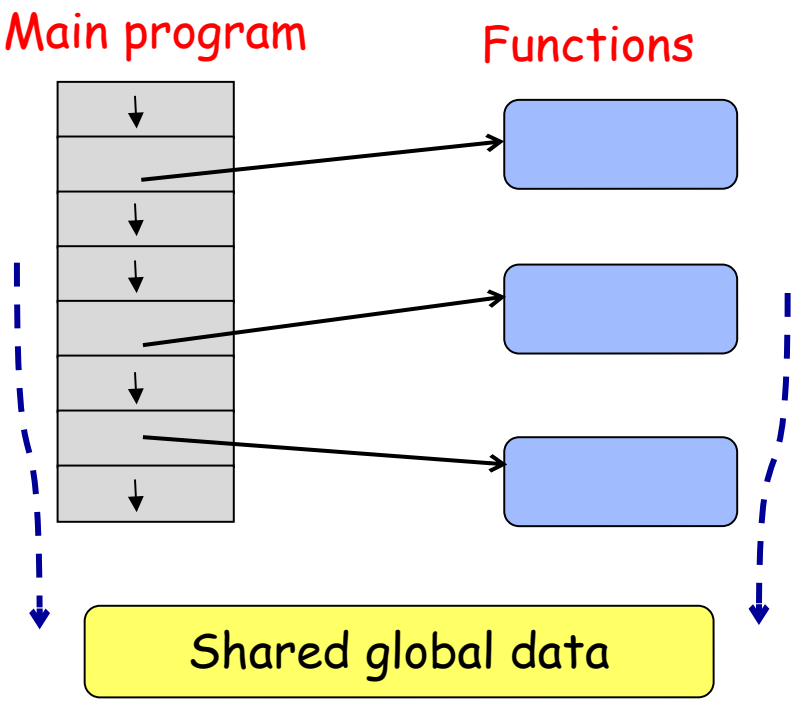
- Program development is based on **models** of real world situations.
- Computer programs are the implementions (coding) of models.

- **Modelling** is the design of a software.
- The followings are the tools for software modelling.
  - **UML diagrams** are used for design of classes.
  - **Flow charts** are used for design of algorithms.

- **Implementation** is the coding with a programming language such as C++.

An Obsolete Technique: Procedural Programming

- In a procedural language such as C or Fortran, the emphasis is on **functions, not objects**.
- A program is divided into **functions.**
- Main program and functions can use shared global data, as well as the passed parameters.



Disadvantages of Procedural Programming

- **Data** is less emphasized, **functions** are more emphasized.

- Procedural programs don't model the real world very well. (The real world does not just consist of functions.)

- To add new data items, all the functions that access the shared data must be **modified,** so that they can access the new items.

The Object-Oriented Approach

- The fundamental idea behind object-oriented programming is :
  - **The real world consists of objects.**

- Thinking in terms of objects, rather than functions, makes the software design easier.

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

- A problem will be easier to understand and handle, if you organize things as **objects**.

Example : University System

A University System software may contain the following entities (objects):

Students have an identification number (ID) and courses attended. They take grades, their GPAs are calculated.

Instructors give courses, they perform some projects, they have some administrative duties.

Courses are given at specific times in a specific classroom. They have a plan, they have a list of enrolled students.

Building Blocks of Object Oriented Programming (OOP)

- **OOP** is a programming technique that organizes software design around data, or objects; instead of functions.
- **Classes** are programmer-defined data types that act as the blueprint for individual objects, attributes and methods.
- **Objects** are instances (variables) of a class created with specifically defined data. Objects can correspond to real-world objects or an abstract entity.
- **Attributes** are datas defined in the class and represent the state of an object. Objects will have data stored in the attributes field.
- **Methods** are functions that are defined inside a class that describe the behaviors of an object. Programmers use methods for re-usability or keeping functionality encapsulated inside one object.

Principles of Object Oriented Programming

- **Encapsulation :**
  - This principle states that all important information (data) is contained inside a class and only some information is exposed.
  - The implementation and data are privately held inside a defined class.
  - Other objects do not have access to this class or the authority to make changes.
  - They are only able to call public functions (methods).
  - This principle provides coding security and avoids unintended data corruption.

- **Inheritance :**
  - Classes can re-use code from other classes.
  - Relationships and sub-classes between objects can be assigned, enabling developers to reuse common codes while still maintaining a unique hierarchy.
  - This principle reduces development time and ensures a higher level of accuracy.

- **Polymorphism :**
  - Objects are designed to share behaviors and they can take on more than one form.
  - The program will determine which meaning or usage is necessary for each execution of that object from a base class, reducing the need to duplicate code.
  - A derived class is then created, which extends the functionality of the base class.
  - This principle allows different types of objects to pass through the same interface.

Advantages of OOP

- The advantages of OOP include the followings :
  - Readability (understandability)
  - Low probability of errors
  - Easy maintenance
  - Modularity
  - Re-usability
  - Productivity
  - Scalability
  - Efficiency

- OOP works very well for complex and large projects that require continous updates and maintenance.
- Examples of such programs include operating systems, compilers, manufacturing and design.

Object Oriented Approach

- Real-world objects have two parts:
  - **Attributes** (Data)
  - **Methods** (Functions)

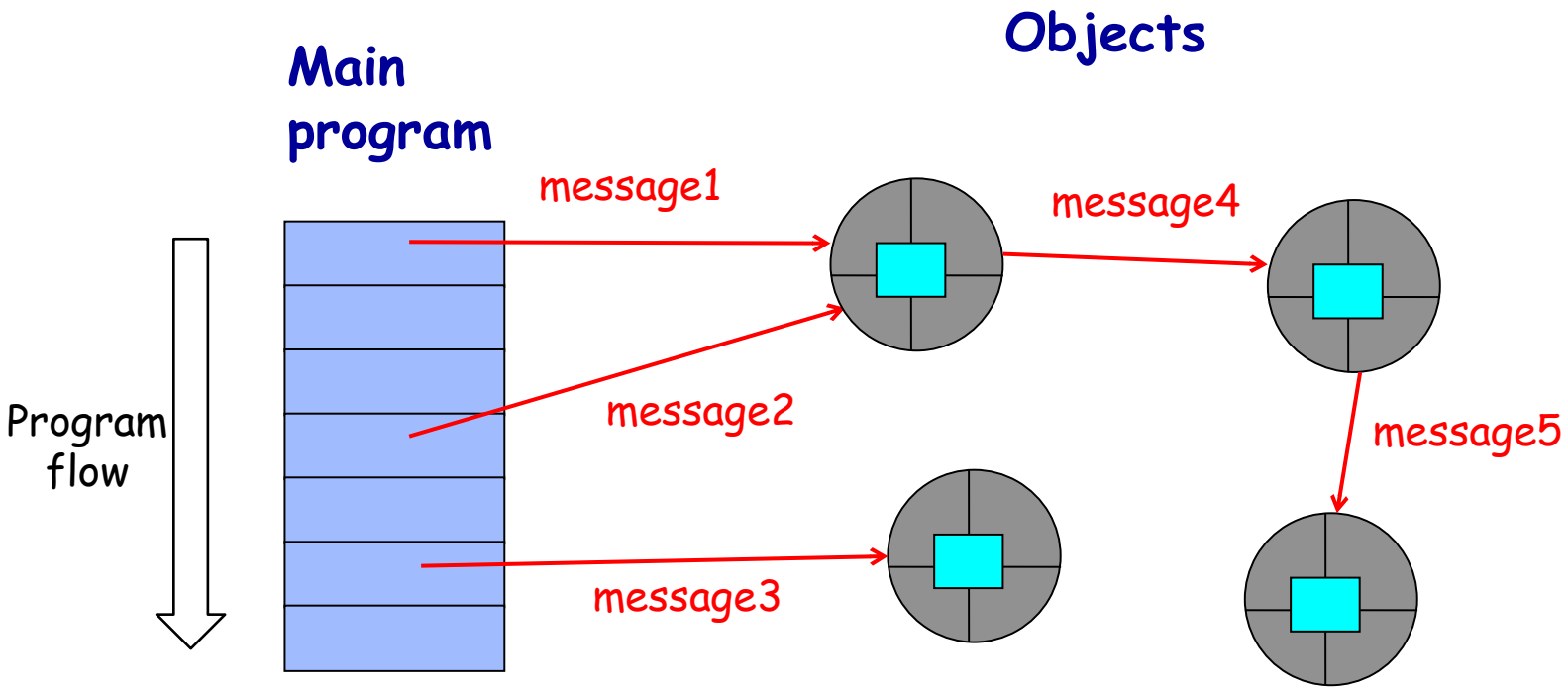
- Software objects correspond to real-world objects.
- Examples of software objects:
  - **Graphics program:** Point, Line, Rectangle, Circle, etc.
  - **Mathematics:** Complex numbers, Matrix
  - **Graphical user interface:** Windows, Menus, Buttons, Toolbars
  - **Data structures:** Arrays, Stacks, Queues, Linked Lists

Object Oriented Approach

- To create software models of real world objects, both *data* and *functions* are combined into a single program entity (Class).
- In OOP, data and its functions are **encapsulated** into a single entity (class).
- A C++ class is a structure declaration similar to a C struct.
- An object is an instance (variable) of a specific class.
- The data of an object can be private, so it cannot be accessed directly.
- The private data can only be changed through its functions (also known as its public interface).
- Classes simplify writing, debugging, and maintaining the program.

Structure of an Object Oriented program

- In OOP, objects combine member data and member functions.
- A C++ program consists of a number of objects that communicate with each other by calling member functions.
- Messages are member function callings of an object with necessary parameter values.



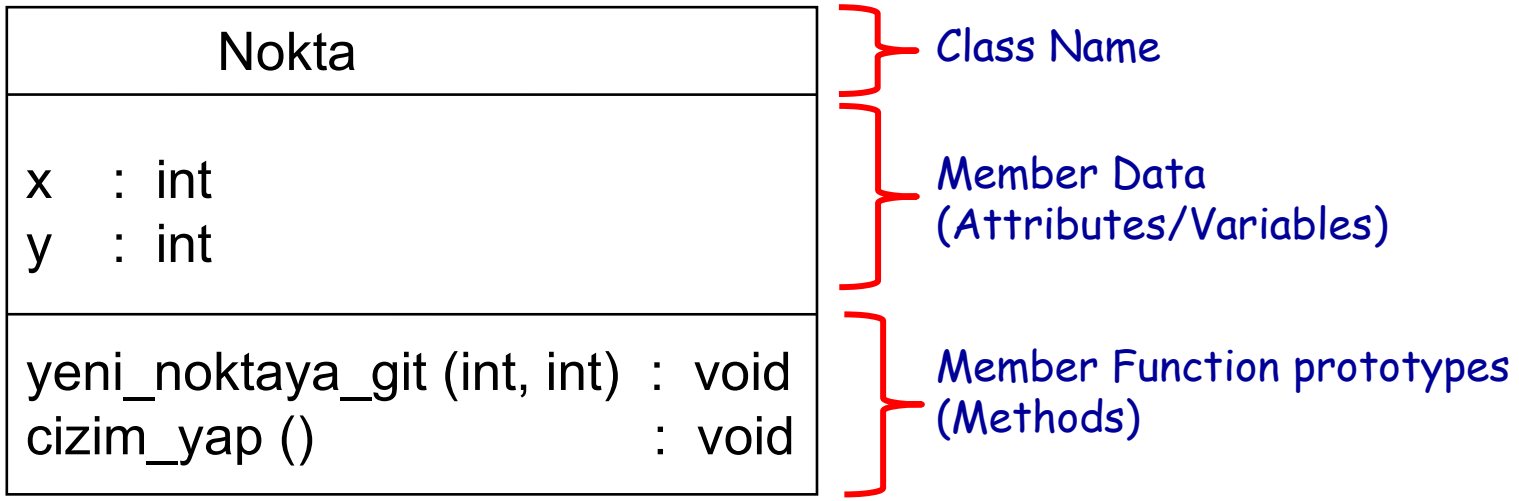
Example:
Point class in a Graphics program

The Point (Nokta) class can be defined with following members.

- **Integer variables (x, y)** : Coordinates of a point.
- **yeni_noktaya_git() function :** Moves the point to a new (x, y) coordinate.
- **cizim_yap() function :** Draws a line from previous point to current mouse point on graphics screen.

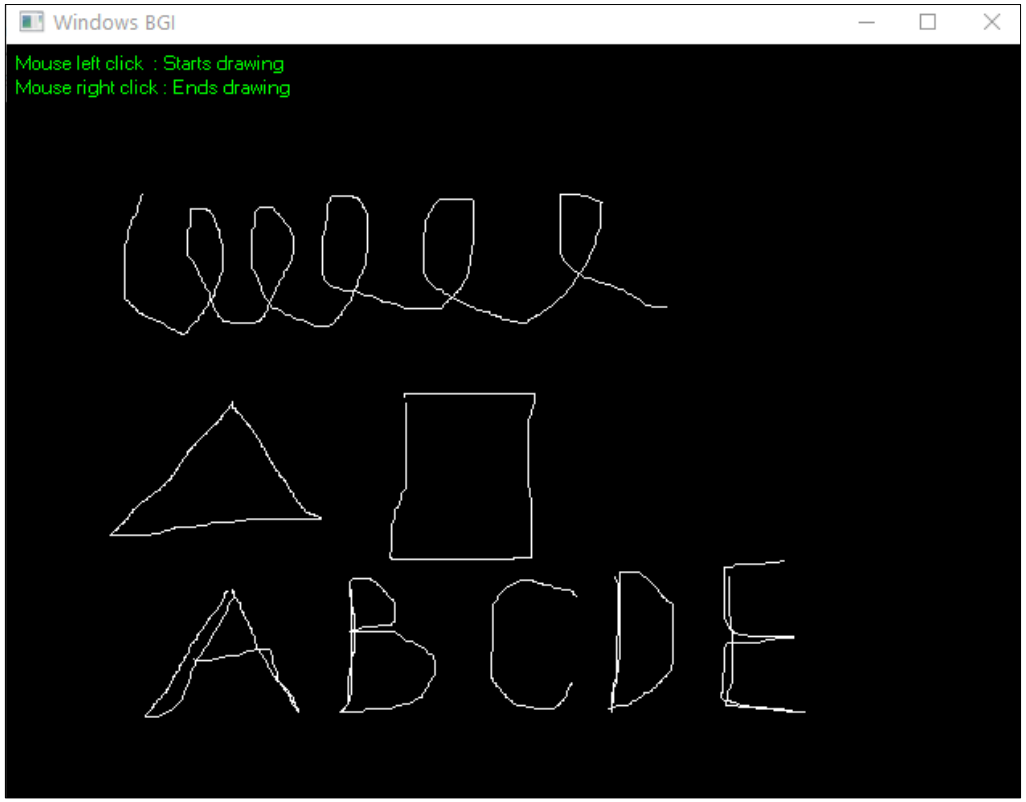
UML class diagram for the Point class

- An **UML (Unified Modeling Language)** class diagram is used as a design tool for modelling of a class.
- It has three sections:
  - Top section : Contains the name of class.
  - Middle section : Contains the declarations of member variables of class.
  - Bottom section : Contains the prototypes of member functions of class.



Example: Program for Curve Drawing with Mouse

- The example program reads mouse clicks and movements to draw continous curves on a graphics window as shown below.
  - User left-clicks the mouse once to start drawing a curve.
  - User moves the mouse to draw a curve on the graphics window.
  - User right-clicks the mouse once to finish drawing a curve.



Program Description

- **Borland Graphics Interface (BGI)** built-in (ready-made) library functions are used.
- BGI library consists of the winbgi.cpp and graphics.h files.
- The Dev-C++ project file (ornek.dev) contains the following files:
  - **winbgi.cpp** library file
  - **graphics.h** library file
  - **cizim.cpp** main program file

- When program is executed, two windows are created automatically.
  - **Console Window** : Can be used for normal program input/output functions such as printf, scanf, etc.
  - **Graphics Window** : Can be used only for graphics-related functions such as initgraph, closegraph, setcolor, settextstyle, outtextxy, line, mousedown, etc.

cizim.cpp File

```
//Project options linker parameter :   -lgdi32
#include "graphics.h"

//Graphics window dimensions
#define WIDTH 640
#define HEIGHT 480

//(continued)
```

Declaration of the Point class

The declaration of Point (Nokta) class should be written outside of main program.

```
class Nokta
{
   int x, y;   // Location coordinates of a point

   public: // Access mode for member functions
   void yeni_noktaya_git(int x_yeni, int y_yeni)
   {
        //Set the location coordinates of point object
        x = x_yeni;
        y = y_yeni;
   }

   void cizim_yap()
   {
        line(x, y, mousecurrentx(), mousecurrenty());
        //Draws a line between old Point location and new mouse location
   }

}; //end of class
```

Main program

Objects (such as the nokta_obj variable) of the Point class can be defined inside main program.

```
int main()
{
   int GraphDriver = 0, GraphMode = 0;
   // Start the graphics window
   initgraph(&GraphDriver, &GraphMode, "", WIDTH, HEIGHT);

   settextstyle(SMALL_FONT, HORIZ_DIR, 5);   // Sets font type and size
   setcolor(GREEN);   //Set color of texts
   outtextxy(5, 5,  "Mouse left click  : Starts drawing");
   outtextxy(5, 20, "Mouse right click : Ends drawing");

   setcolor(WHITE);   //Set color of pen
   Nokta  nokta_obj;      //Definition of a Point class object
   bool CIZIM_DURUMU = false; //Boolean flag variable (used to  enable/disable drawing)
```

Main program (continued)

```
while (true) // Endless loop
{
  if (mousedown() == true)  {
     if (whichmousebutton() == LEFT_BUTTON)
     {
        CIZIM_DURUMU = true; //Drawing is now enabled
        nokta_obj.yeni_noktaya_git(mouseclickx(), mouseclicky());
        //set new location for object
      }
      else
        CIZIM_DURUMU = false; //Drawing is now disabled
  }

  if (CIZIM_DURUMU == true)
  {
     nokta_obj.cizim_yap(); //Call drawing function of object
     nokta_obj.yeni_noktaya_git(mousecurrentx(), mousecurrenty());
     //set new location for object
  }

} //end of while

} //end of main
```