Selsoft Academy

OOP from Different Language Perspectives: C++, Java, C#

Akın Kaldıroğlu

13 Septemper 2017

Agenda

- Discussing the choices of different languages to implement the mechanisms of object-oriented programming.

- Languages discussed are C++, Java, C# in historical order.

- I tried to come up with the languages that reflect the development of our understanding of OOP since 80's.

Several Points - I

- I want to proceed in this talk rather in a more interactive manner so please feel free to join the talk.
- Different languages have different cultures or conventions including naming and representing things.
  - Main convention in this work is that of Java although I try to observe others as much as I know.

Several Points

- Due to changes and improvements in different versions of languages I mentioned here I may make mistakes about their features so please don't hesitate to correct or suggest an alternative regarding the issue.

Theory

Software and Objects

- An Aristotalian deduction:
  - Software systems are simulations of the World,
  - World can be seen as totality of objects (or facts?),
  - So software systems can be seen as simulations of objects of the World.
  - **Constructing software systems using objects is the most natural way to develop software.**

What is Object? - I

- English word **object** stems from Latin word *obiectus* (or *objectus*), passive participle of *obicio*.
  - *Obicio* in Latin means “throw or put in front of or before”
    - *ob* means towards, against, *jacere* means to throw, to put
      - object vs. inject!
- Object is anything against the subject, which is the mind,
  - Object is anything the mind percieves or think.

What is Object? - II

- In archaic Greek object is *αντικείμενο* (*antikeimena*), meaning *karşıdaki şey*

- In Osmanlıca, it is *müteallak/mütekabil*
  - *Taalluk*: İlgisi olma, ilgi, bağlantı
  - *Tekabül*: Karşılık olma

- In modern Turkish, it is *nesne*, *ne ise* (*nesene, nim érse*)

What is Object? - III

- Philosophically, object is bundle of qualities/ properties.
- So what are those qualities?

Categories

Aristo'nun 10 Kategorisi			
Aristo	**English**	**Turkish**	**Örnek**
Ti esti, ousia	Substance	Cevher/Töz	İnsan
Poson	Quantity	Nicelik	1.75 boyunda
Poion	Quality	Nitelik	Esmer
Pros ti	Relation	Görelik	Baba
Pou	Place	Mekan	Ormanda
Pote	Date	Zaman	Dün
Keisthein	Posture	Durum	Ayakta
Echein	State	Pozisyon	Spor kiyafetleriyle
Poitein	Action	Etki	Zıplıyor
Paschein	Passion	Edilgi	Yoruluyor

- According to Aristotle, there are 10 types of things.
  - Substance can exist as its own and all others can only exist as long as they are attributed to or said of a substance.

Aristo'nun 10 Kategorisi			
Turkish	**Örnek**	**Nesne**	**Sorular**
Cevher/Töz	İnsan	Nesnenin tipi	O nedir?
Nicelik	1.75 boyunda	Nesnenin niceliksel özellikleri	O kaç ya da ne kadar ...dır? Ya da Onun ...sı kaçtır ya da ne kadardır?
Nitelik	Esmer	Nesnenin niteliksel özellikleri	O .. açısından nasıldır? Onun ..sı nasıldır?
Görelik	Baba	Nesnenin diğer nesnelerle ilişkileri	Onun diğer şu nesne ile ilişkisi nedir?
Mekan	Ormanda	Nesnenin mekansal özelliği	Nesne nerededir?
Zaman	Dün	Nesnenin zamansal özelliği	Nesne ne zaman ... olmuştur?
Durum	Ayakta	Nesnenin o anki durumu	Nesne şu anda ne durumda/haldedir?
Pozisyon	Spor kiyafetleriyle	Nesnenin sahip oldukları	Nesne nelere sahiptir?
Etki	Zıplıyor	Nesnenin yaptıkları	Nesne neler yapar?
Edilgi	Yoruluyor	Diğer nesnelerin bu nesneye karşı yaptıkları.	Nesne hangi olayları (event) alır?

Properties of Objects

- So a substance may have some properties expressed in terms of countable, uncountable, temporal and positional qualities,
- It holds some relationships with other substances,
- It has some actions or behaviors and
- It receives some events.

Substance of Object

- It is substance that holds all properties and behaviors that can be said of the objects of that substance.

- Substance is the form or archetype of object.

- Substance is the common structure of the objects of the same form.

Relationships Among Substances

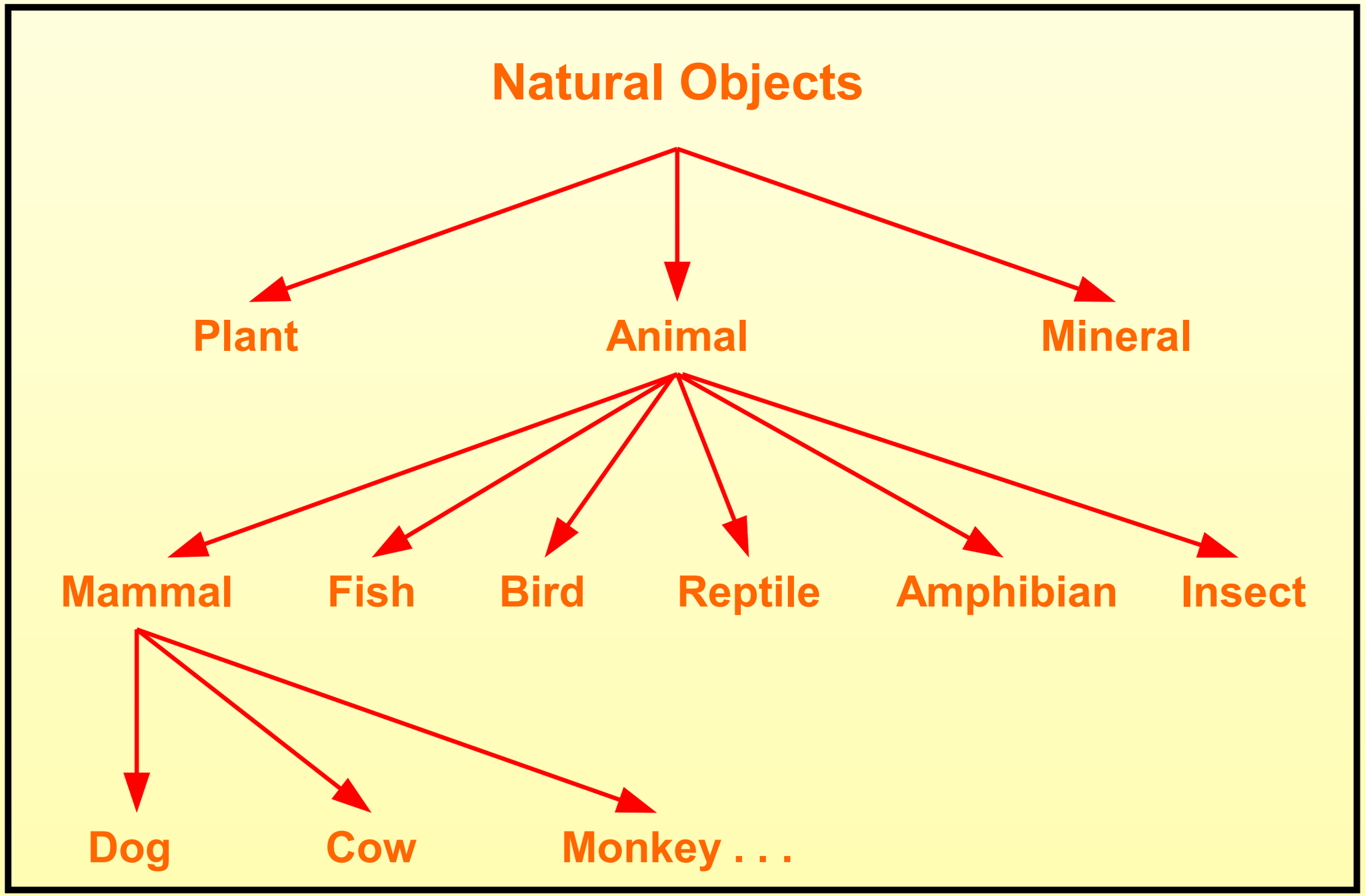
- Different substances can have relationships among themselves.
- Substances may be attributed to other substances.
- Substances may have hierarchies.

Association Among Substances

- Substances may be attributed to or associated with other substances.
  - A substance may be a quality of another.
- When a substance is attributed to another substance, relationship among them seem to be more contact point-oriented i.e. interactions happen on specific contact points on substances leaving unrelated aspects untouched.

Substance Hierarchies

- Substances may have hierarchies.
- Substances lower in the hierarchy share the same qualities of the substances in the upper.
  - This is called inheritance


Natural Objects
Plant
Animal
Mineral
Mammal
Fish
Bird
Reptile
Amphibian
Insect
Dog
Cow
Monkey . . .

Transition from Theory

Programming Languages

- Programming languages are for developing software systems.

- Programming languages that have mechanisms to apply the ontology mentioned before are called object-oriented.

Object of Software - I

- In terms of *object of the software*,
  - qualities are expressed in terms of different types of data,
  - actions and events are expressed in terms of functions or methods.
- Each quality of the object is called **data member**, **field** or **property**.
- Totality of the methods that can be called on an object is called **interface**.

Encapsulation & Information Hiding

- Packaging a substance with its properties and methods is called **encapsulation**.
  - An encapsulated substance turns to a type in OOPLs.
- Encapsulations mostly have a complementary mechanism called **information hiding** to abstract away from others its complexities regarding its inner workings.

Programming Abstractions - I

- OOPLs mainly abstract two things in a type:

- Data abstraction: It abstracts away the inner complexities of abstract data types.

- Process abstraction: It abstracts away how data is processed.

Programming Abstractions - II

- OOPLs should also have mechanisms to express relationships among types i.e. associations and inheritance.

- Moreover OOPLs should provide access control mechanisms to enforce information hiding.

Good Software

- Good software systems are those that
  - do what is expected by their users correctly,
  - do it using a reasonable amount of resources,
  - are easy to understand and maintain which is mainly modifiying by adding new functionalities.
- Which one do you thing is the most difficult? Can you order them?

Coupling and Cohesion

- To create easy-to-understand-and-maintain software systems we need to make types highly-cohesive and lowly-coupled.

- So as the types of the software should correctly depict the substances of the real world

Implementation

Encapsulation

Encapsulation in OOPLs

- C++, Java and C# has `class` keyword to create a type or encapsulation.
  - There are other types of encapsulation in those languages but for now let's focus on classes as main encapsulation mechanism.
- Go has `struct` keyword instead of `class` to do the same thing.

Car.cpp

```
using namespace std;
class Car{
public:
    string make;
    string model;
    string year;
    unsigned int speed;
    unsigned int distance;

    void go(int newDistance) {
        distance += newDistance;
    }

    void accelerate(int newSpeed) {
        speed = newSpeed;
    }

    void stop() {
        speed = 0;
    }

    string getInfo() {
        return "Car Info: " + year + " " + make + " " + model + ". Distance: " +
        to_string(distance) + " km. and traveling at " + to_string(speed )
        + " kmph.";
    }

   // setters/getters
};
```

Car.java

```java
package org.javaturk.oopl.java;

public class Car{
    private String make;
    private String model;
    private String year;
    private int speed;
    private int distance;

    public void go(int newDistance) {
        distance += newDistance;
    }

    public void accelerate(int newSpeed) {
        speed = newSpeed;
    }

    public void stop() {
        speed = 0;
    }

    public String getInfo() {
        return "Car Info: " + year + " " + make + " " + model + ". Distance: " +
        distance + " km. and traveling at " + speed + " kmph.";
    }

    // setters/getters
}
```

Car.cs

```
using System;
namespace car{
    public class Car{
        string make;
        string model;
        string year;
        int speed;
        int distance;

        public void go (int newDistance){
            distance += newDistance;
        }

        public void accelerate (int newSpeed){
            speed = newSpeed;
        }

        public void stop (){
            speed = 0;
        }

        public string getInfo (){
            return "Car Info: " + year + " " + make + " " + model + ". Distance: "
           + distance + " km. and traveling at " + speed + " kmph.";
        }

       // setters/getters
    }
}
```

Access Control for Classes - I

- How do you can control the access to a class?
- It is important to hide some classes from client code so that they dont show up in the API.
- That’s information hiding at class level!
- In C++ there is no way to control the access of a class through an access modifier.

Access Control for Classes - II

- In Java if a class is not declared *public*, nobody outside its package can access it.
  - This is called *default* or *friendly* accessibility.
- In C#, you can declare a class either *internal* or *public*.
  - A class declared *internal*, which is the default case, can not be accessed outside its current assembly.

Access Control for Classes - III

- The access control mechanism in Java is definitely an improvement over C++.

- Apparently C# takes over this approach with a small modification:

- In Java granularity of the access control mechanism is finer than that is in C#.

- Java prefers package/namespace level control while C# prefers assembly level control.

Access Control for Members - I

- For the member level access control, in case of no modifier used, all members in C++ and C# are **`private`** but in Java they are *package-accessable*.

- That means in default case C++ and C# behaves more strictly than Java.

- C# provides a richer set of access control keywords and thus granularity.

Access Control for Members - II

- C++ and Java provides three different keywords for the access control of members:
  - **`public`**
  - **`protected`**
  - **`private`**
- C++ provides three levels of access while Java provides four levels with the omitted use of any keywords.

Access Control for Members - III

- Although **`public`** and **`private`** means the same thing in C++ and Java, **`protected`** provides a little bit larger access in Java.

  - **`protected`** in both allows access from child/sub/derived classes,

- **`protected`** in Java allows access from within the same package as well.

- So in Java, there is no way to hide a member from the package but making it accessible for children!

Access Control for Members - IV

- Like C++, C# does not provide package/namespace level access control but provides assembly level.

- Assembly level access control is achieved by another keyword **`internal`**
  - It is larger access level than default/package access level in Java.

- For Java if it is not for outside of a package that means it is not for anybody unless from within a child!

Access Control for Members - V

- **`protected`** in C# behaves exactly as in C++ whereas in Java it also allows access from within the package.
- Moreover C# allows **`internal`** and **`protected`** keywords to be used together to let both current assembly and child classes to access.
  - This case is closer to protected of Java except C# prefers current assembly instead of package.

Scope

- Java and C# don’t allow any global variable or function.

- All members must be encapsulated and scoped in a class.

- But C++ allows global variables and functions that have no class scope.

Global.h & main.cpp

```
#ifndef Global_h
#define Global_h

unsigned int top_speed = 200;

void wash_car(Car car){
    cout << "Washing the car: " + car.getMake() + " " +
            car.getModel() + " of " + car.getYear() << endl;
}

#endif /* Global_h */
```

```
// main.cpp
Car car1;
car1.setMake("Mercedes");
car1.setModel("C200");
car1.setYear("2017");
car1.setDistance(0);
car1.setSpeed(0);
car1.setSpeed(top_speed);

cout << car1.getInfo() << "\n" << endl;

wash_car(car1);
```

Encapsulation Problem with C++

- That means in C++ you can write code without actually having a class or even though you write classes you can still have practically any piece of code outside of any class.

- That leads to pure procedural programming!

Other Types of Encapsulation

Struct & Enum Encapsulations

- Although C++ and C# have another type of encapsulation called **`struct`** it does not add much of value to OOP.

- Similarly C++, Java and C# has **`enum`** types as a specific type of encapsulation.

- Since they are kind of helper encapsulation types we don't focus on them.

Process Encapsulation - I

- Another type of encapsulation packages only interfaces of methods.

- It is process encapsulation with no implementation.

- It provides only encapsulaton of method interfaces and does not provide any implementation at all.

Process Encapsulation - II

- Java and C# has `interface` keyword to create such an encapsulation.

- C++ does not provide a specific mechanism in the language to specify define interfaces.

- Abstract methods and classes are used to this end.

- But the way of Java and C# is more elegant and enhancing OO understanding and practice.

Interfaces in Java & C#

- With version 8, Java started allowing interfaces to have implementations too.

- This is done due to specific needs Java SE 8 faces when enhancing the language with some functional programming features.

- C# they are still pure, uncontaminated interfaces!

  - But it looks like C# 8.0 will have default methods in interfaces.

Cutter.java, Cutter.cs & Cutter.h

```java
public interface Cutter {

    public void cut();

}
```

```cs
using System;
namespace Interfaces
{
    public interface Cutter
    {
        void cut ();
    }
}
```

```h
#ifndef Cutter_h
#define Cutter_h

class Cutter{

    public:

    virtual void cut() = 0;
};
#endif /* Cutter_h */
```

Inheritance

Inheritance - I

- OOPLs provide mechanisms to create hierarchies among encapsulations.
  - It is called inheritance.
- Hierachies provide generalization-specialization relationshsip among types.
- Inheritance is also called is-a relationship.

Inheritance - II

- By inheritance the types that are lower in the hierarchy may inherit two things from their parents:
  - Data abstractions
  - Process abstractions
- Although most of the time both of them are considered to be inherited by class inheritance it is not the case when the parent is an interface.

Inheritance Notation

- To create an inheritace relationship C++ and C# uses “:” notation while Java uses **`extends`** keywords.

F1Car.java

```java
public class F1Car extends Car{

    private String pilot;

    @Override
    public void accelerate(int newSpeed) {
        System.out.println("Faster acceleration!");
        speed = newSpeed;
    }

    @Override
    public String getInfo() {
        String info = super.getInfo();
        return info + "Pilot is " + pilot;
    }

    public String getPilot() {
        return pilot;
    }

    public void setPilot(String pilot) {
        this.pilot = pilot;
    }
}
```

F1Car.h

```
class F1Car : public Car{

    private:
    string pilot;

    public:
    string getPilot(){
        return pilot;
    }

    void setPilot(string pilot){
        this->pilot = pilot;
    }

    void accelerate(int newSpeed) {
        cout << "Faster acceleration!" << endl;
        speed = newSpeed;
    }

    string getInfo() {
     return "F1Car Info: " + year + " " + make + " " + model + ". Distance: "
                + to_string(distance) + " km. and traveling at "
                + to_string(speed ) + " kmph and driven by " + pilot + ".";
    }
};
```

Polymorphic References

- C++, Java and C# supports polymorphic references.
- A reference of a parent type can refer to any reference of its child types.
- In case of C++ those references are pointers whereas in case of Java and C# they are only references.

CutterFactory.cs & CutterFactory.h

```
public class CutterFactory
{
    private static Random random =
                        new Random ();

    public static Cutter createCutter ()
    {
      Cutter cutter = null;

      int i = (int)(3 * random.NextDouble ());

      switch (i) {
      case 0:
          cutter = new Actor ();
          break;
      case 1:
          cutter = new Barber ();
          break;
      case 2:
          cutter = new Butcher ();
          break;
      }
      return cutter;

    }
}
```

```
class CutterFactory {

   public :

   Cutter * createCutter() {
      Cutter *cutter = NULL;

      int i = (int) (3 * (double)
                rand() / (RAND_MAX));

      switch (i) {
          case 0:
                cutter = new Actor;
                break;
          case 1:
                cutter = new Barber;
                break;
          case 2:
                cutter = new Butcher;
                break;
      }
      return cutter;
    }
};
```

Overriding

- When a child class inherits from its parent class it can override methods it inherits.

- This is called **overriding** and the methods that can be overridden is called **polymorphic**.

Polymorphic Methods - I

- In Java all non-static methods are in default polymorphic.

- This is not the case with C++ and C#.
  - In default the methods in C++ and C# are not polymorphic.

- In order to have polymorphic methods in C++ they need to be declared **`virtual`**.

F1Car.h

```
class F1Car : public Car{

    private:
    string pilot;

    public:
    string getPilot(){
        return pilot;
    }

    void setPilot(string pilot){
        this->pilot = pilot;
    }

    void accelerate(int newSpeed) {
        cout << "Faster acceleration!" << endl;
        speed = newSpeed;
    }

    string getInfo() {
     return "F1Car Info: " + year + " " + make + " " + model + ". Distance: "
                + to_string(distance) + " km. and traveling at "
                + to_string(speed ) + " kmph and driven by " + pilot + ".";
    }
};
```

F1Car.h

```
class F1Car : public Car{

    private:
    string pilot;

    public:
    string getPilot(){
        return pilot;
    }

    void setPilot(string pilot){
        this->pilot = pilot;
    }

    string getInfo() {
     return "F1Car Info: " + year + " " + make + " " + model + ". Distance: "
                + to_string(distance) + " km. and traveling at "
                + to_string(speed ) + " kmph and driven by " + pilot + ".";
    }
};
```

Polymorphic Methods - II

- In default the methods in C# are **`sealed`** and need to be declared by **`virtual`** keyword preceeding it to be polymorphic.

- This main difference between Java's and C++/C#'s methods I believe stresses the mind sets between Java and these two languages.

Final/Sealed Classes and Methods

- Java provides **`final`** *classes* so that they may not be extended.

- Java also provides **`final`** *methods* so that they may not be overriden.

- C# uses **`sealed`** keyword to provide the same thing.

Multiple Inheritance - I

- While C++ allows a class to inherit from more than one class Java and C# don't.

- Allowing multiple class inheritance causes infamous diamond problem.

- C++ mostly solves this problem by enforcing overriding the method that causes this.

Multiple Inheritance - II

- Although Java and C# don't provide multiple class inheritance, they do provide multiple inheritance when only interface not the implementation is inherited.
  - So using interfaces is another way to implement multiple inheritance in Java and C#.
  - A class can inherit from multiple interfaces.
- Due to radically changed nature of interfaces in Java SE 8, infamous diamond problem occurs.

To Sum Up

C++ - I

- C++ is a hybrid langauge that allows both pure procedural and pure object-oriented programming.

- I guess this hybrid nature of C++ is due to mainly two facts:

- C++ hasn’t been designed from scratch, designed to be a better C, so it is like an add-on to C.

- C++ is designed for any possible programming tasks.

C++ - II

- This makes C++ very applicable and less-verbose for lower-level tasks that does not need to have objects.

- On the other hand the procedural aspects of C++ can be considered a way to flee from objects.

C++, Java and C#

- Despite all its problems C++ is a huge improvement with its object mechanisms to rather a slow transition to OOP while being still best fit for procedural style.

- IMHO Java was the language to complete this transition with lots of prunning and corrections.

- And C# was the Microsoftian way of Java.

Java and C#

- In this sense C# was not a revolutionary language but more like a Java-like language with Microsoft style.

- C# reflects every single pragmatism of Microsoft in that frequent releases, rich feature sets and totally tool oriented programming through skillful VS IDE to let almost anybody to write C# code.

Chain of Languages

- When I think of the chain of languages in general purpose and large-scale software development it goes as C/C++, Java and C#.

- There are some alternatives such as Python and Go.

- But I think neither Pyhton nor Go will become a language for mainstream development although they may be good at specific aspects of enterprise development.

Object-Oriented Development - I

- Developing software is hard.
- Developing OO software is still hard.
  - That's because the hardest aspect of OO development is to find the objects themselves!
- Developing easily understandable and changeable OO software is even much harder.

Object-Oriented Development - II

- In daily life we mentally construct lots of new abstractions and revise the existing ones continuously.

- But in software world, managing such a dynamic change is almost impossible.

- OOPLs provide better ways to partition the software systems into logical units making the system more managable and intelligable although different langauges chooses different ways.

Resources

- **Concept of Programming Languages** 11th ed., Global Ed. Sabesta, R. W., Pearson, 2016
- **Object-Oriented Thought Process** 3rd ed., Weisfeld, M., Addison-Wesley, 2009
- **Touch of Class**, Meyer, B., Springer, 2009

Dinlediğiniz için teşekkür ederim.

Bu sunuma

http://www.javaturk.org

adresinden ulaşabilirsiniz.