Object Oriented Programming With C++

Textbook/ References

1. Teach Yourself C++, 3rd Edition, Herbert Schildt.

2. http://www.cplusplus.com/doc/tutorial/

References:

1. C++ How to Program, 4th Edition, Deitel and Deitel.

2. The C++ Programming Language, Special 3rd Edition, Bjarne Stroustrup

3. Thinking in C++, Volume One, 2nd Edition. Bruce Eckel. Downloadable from http://www.BruceEckel.com

4. The C++ Programming Language by Bjarne Stroustrup

5. C++ Primer Plus, 5th Edition by Stephen Prata

6. Essential C++ (The C++ In-depth Series) by Stanley B. Lippman

Difference between procedural programming and OOP

Let us begin with a story of two programmers:

Difference between procedural programming and OOP

Let us begin with a story of two programmers:

Difference between procedural programming and OOP

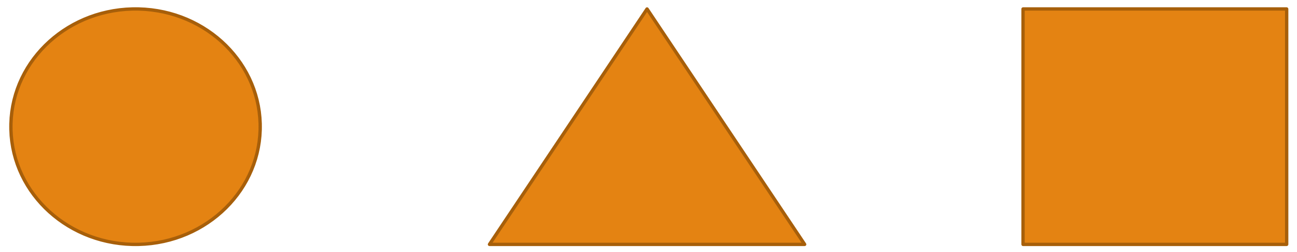
Let us begin with a story of two programmers:

Difference between procedural programming and OOP

One day they both were given some specifications and told to build an application:

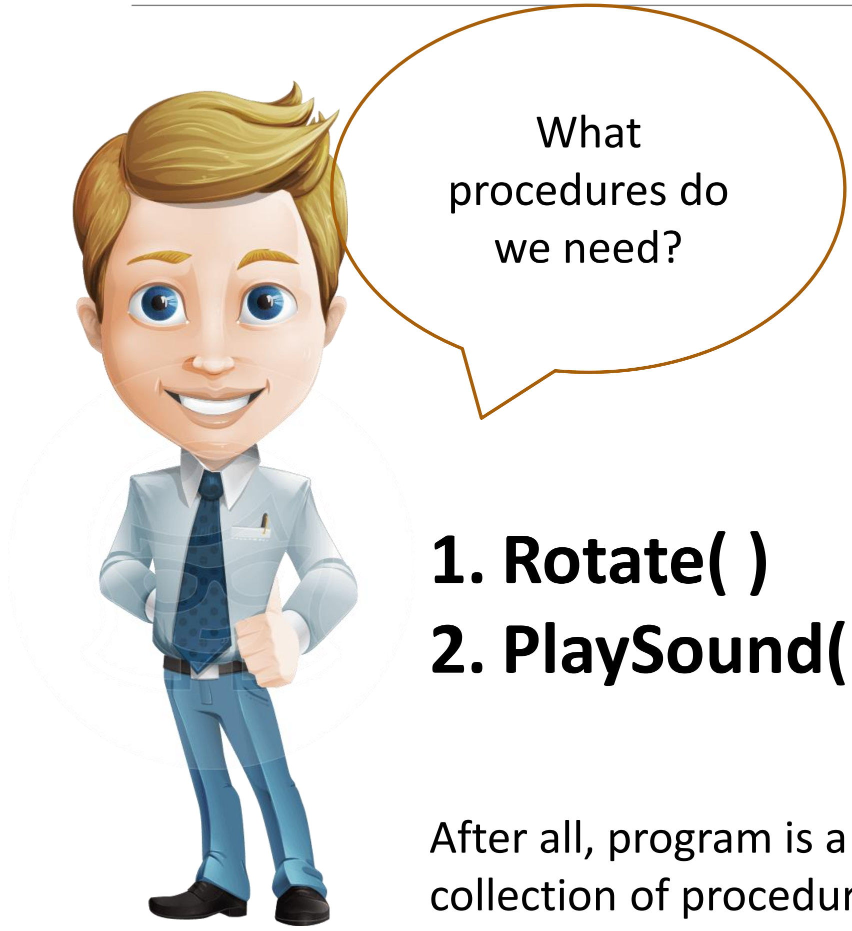
The specifications:

There will be shapes: **Circle, Triangle and Square**



When the user clicks on a shape, the shape will be rotated clockwise 360 degree and will play an wav sound file.

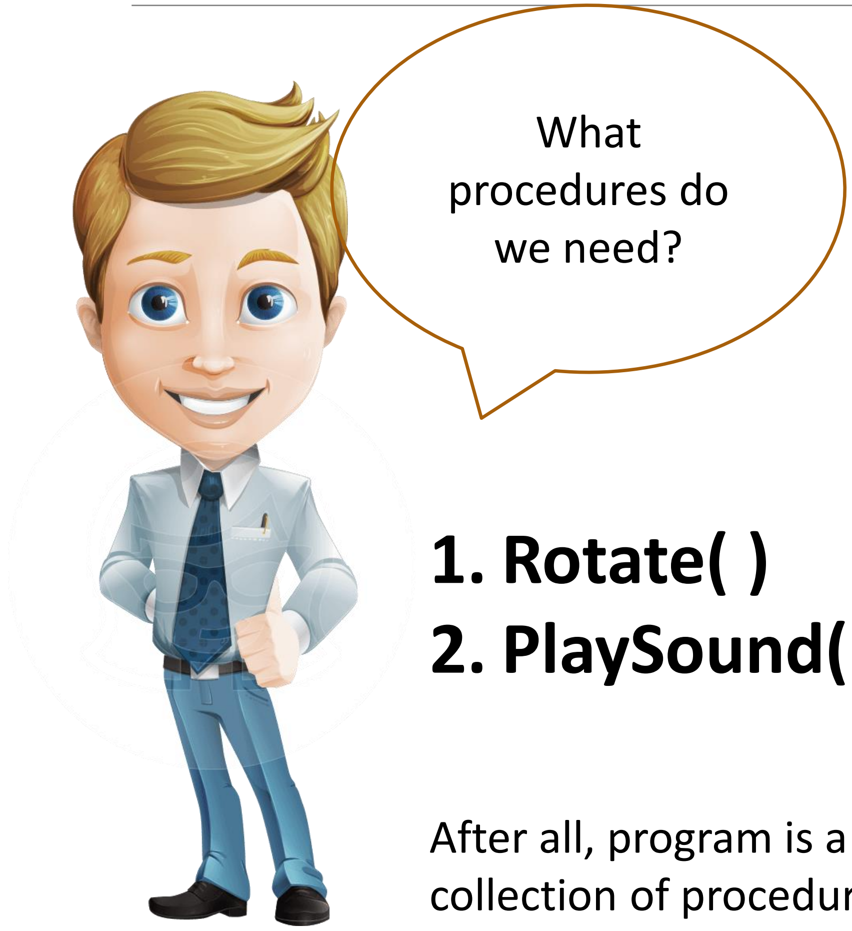
Difference between procedural programming and OOP



1. **Rotate( )**
2. **PlaySound( )**

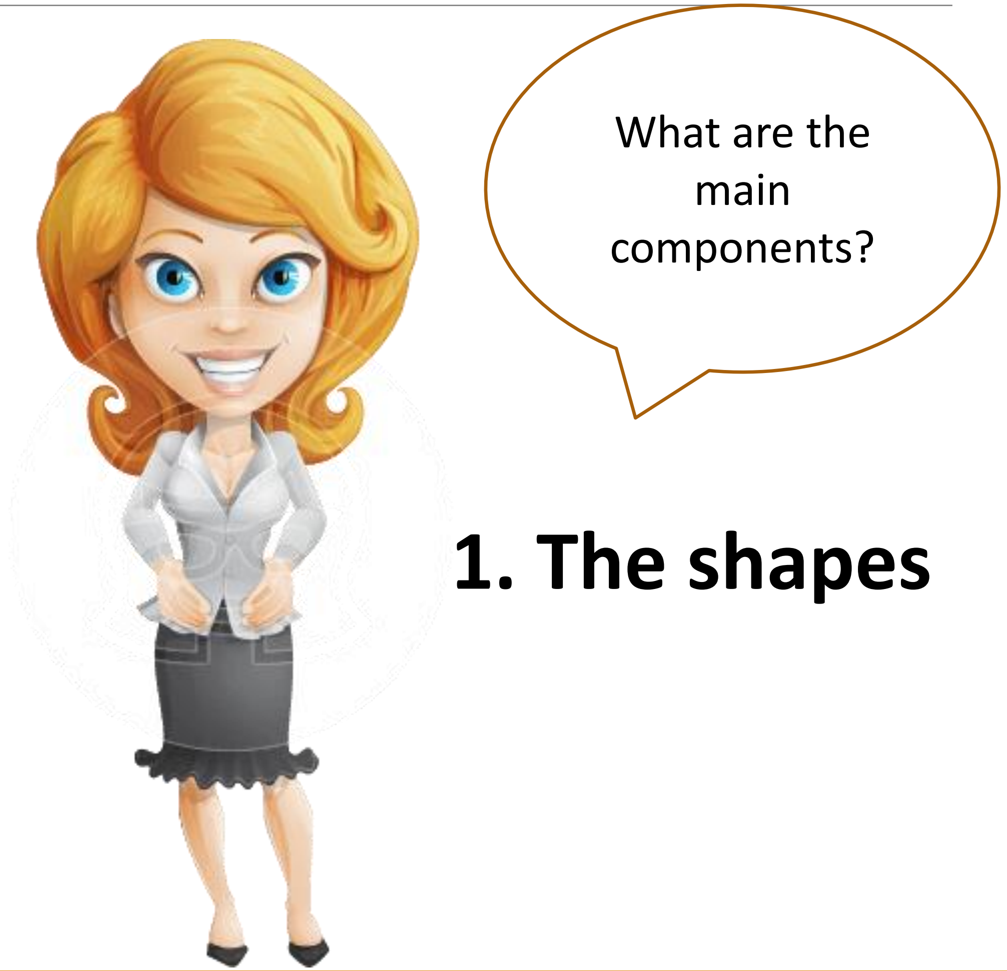
After all, program is a collection of procedures.

Difference between procedural programming and OOP



1. Rotate( )
2. PlaySound( )

After all, program is a collection of procedures.



1. The shapes

Difference between procedural programming and OOP

PROCEDURAL PROGRAMMING

Sameer wrote ROTATE and SOUND in NO time

```
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound( )
{
//play the WAV file
}
```

OBJECT ORIENTED PROGRAMMING

Tania wrote a class for three shapes

```
Class Square
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound( )
{
//play the WAV file
}
```

```
Class Circle
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound( )
{
//play the WAV file
}
```

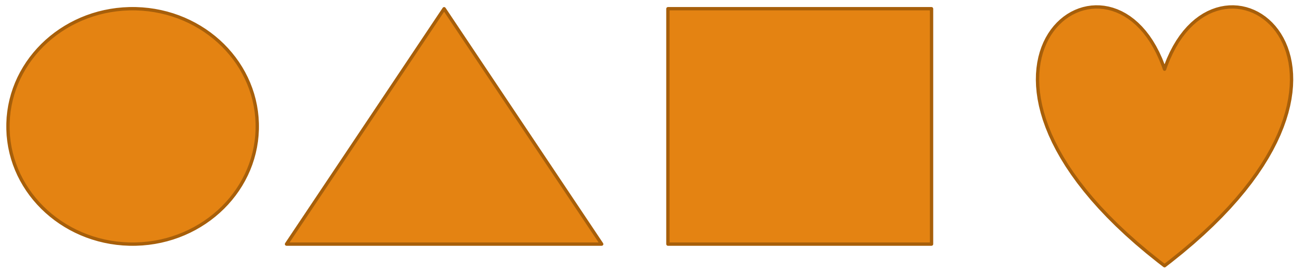
```
Class Triangle
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound( )
{
//play the WAV file
}
```

Difference between procedural programming and OOP

But the manager said: “Wait, there is a new specification..”

A new shape is added: Heart shape along with other shapes

When the user will click on Heart shape it will also rotate 360 degrees
But will play an mp3 file.



Difference between procedural programming and OOP

PROCEDURAL PROGRAMMING

```
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound(ShapeNum)
{
//if the shape is not heart then play .wav file
//else play .mp3 file
}
```

Difference between procedural programming and OOP

PROCEDURAL PROGRAMMING

```
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound(ShapeNum)
{
//if the shape is not heart then play .wav file
//else play .mp3 file
}
```

OBJECT ORIENTED PROGRAMMING

Difference between procedural programming and OOP

PROCEDURAL PROGRAMMING

```
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound(ShapeNum)
{
//if the shape is not heart then play .wav file
//else play .mp3 file
}
```

OBJECT ORIENTED PROGRAMMING

```
Class Square
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound( )
{
//play the .wav file
}
```

```
Class Circle
Rotate()
```

```
Class Heart
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound( )
{
//play the .mp3 file
}
```

```
Class Triangle
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound( )
{
//play the .wav file
}
```

Difference between procedural programming and OOP

Sameer had to make changes in the previous code and will have to do so..

Every time the specification changes

On the contrary, Tania just wrote a new class without making changes to the previously written code

Difference between procedural programming and OOP

PROCEDURAL PROGRAMMING

```
Rotate()
{
//make the shape rotate 360
degrees
}
PlaySound(ShapeNum)
{
//if the shape is not heart then
play .wav file
//else play .mp3 file
}
```

1. Lots of code is effected

2. Previously written code was changed

3. What if the specification changes again??

Like:

For Heart: rotate the shape 180 degree, not 360 degree..

Difference between procedural programming and OOP

PROCEDURAL PROGRAMMING

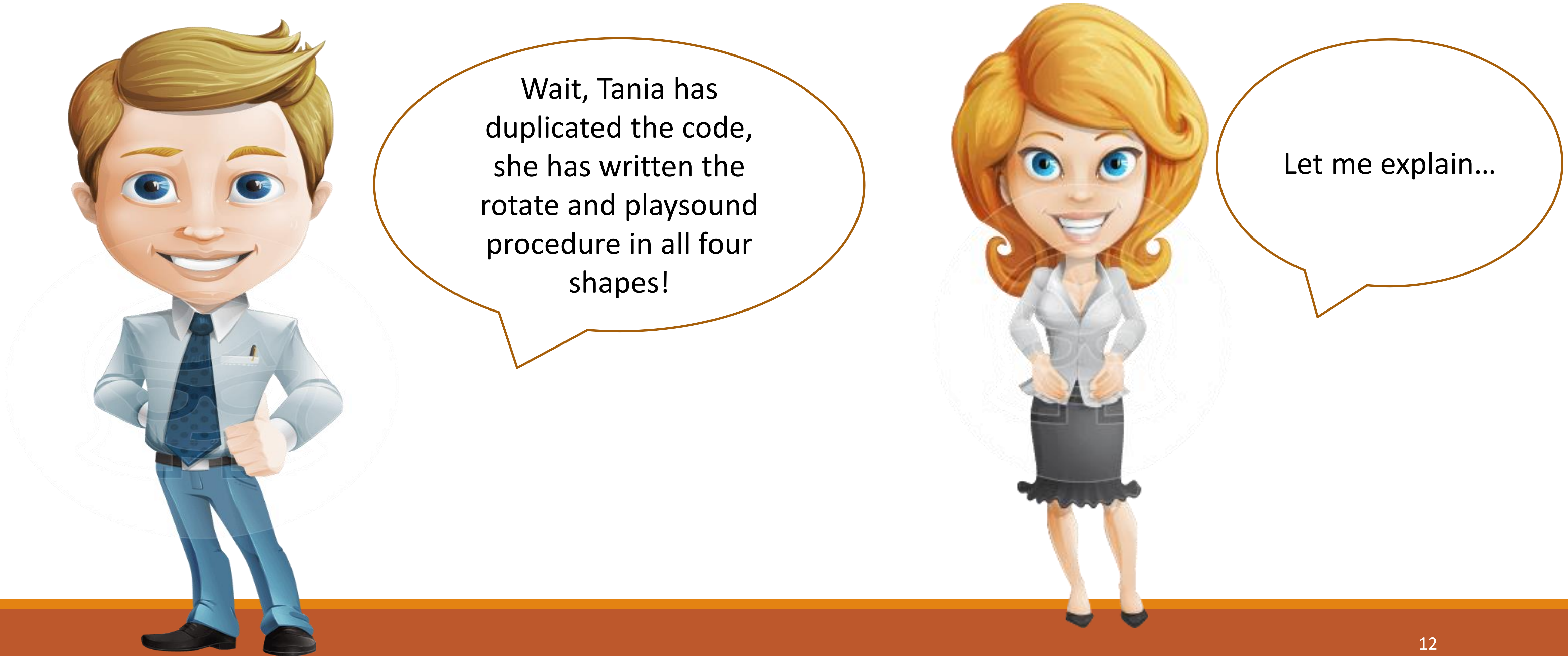
```
Rotate(ShapeNum)
{
//if the shape is not heart rrotate
360 degree
//else make the shape rotate
degrees
}
PlaySound(ShapeNum)
{
//if the shape is not heart the
.wav file
//else play .mp3 file
}
```

OBJECT ORIENTED PROGRAMMING

```
Class Heart
Rotate()
          e shape rotate 180
          ( )
          e .mp3 file
}
```

Which approach is more efficient? Changing one class or editing the entire file!!

Difference between procedural programming and OOP


Wait, Tania has duplicated the code, she has written the rotate and playsound procedure in all four shapes!
Let me explain…

Object Oriented Programming

```
Class Square
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound( )
{
//play the .wav file
}
```

```
Class Circle
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound( )
{
//play the WAV file
}
```

```
Class Triangle
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound( )
{
//play the .wav file
}
```

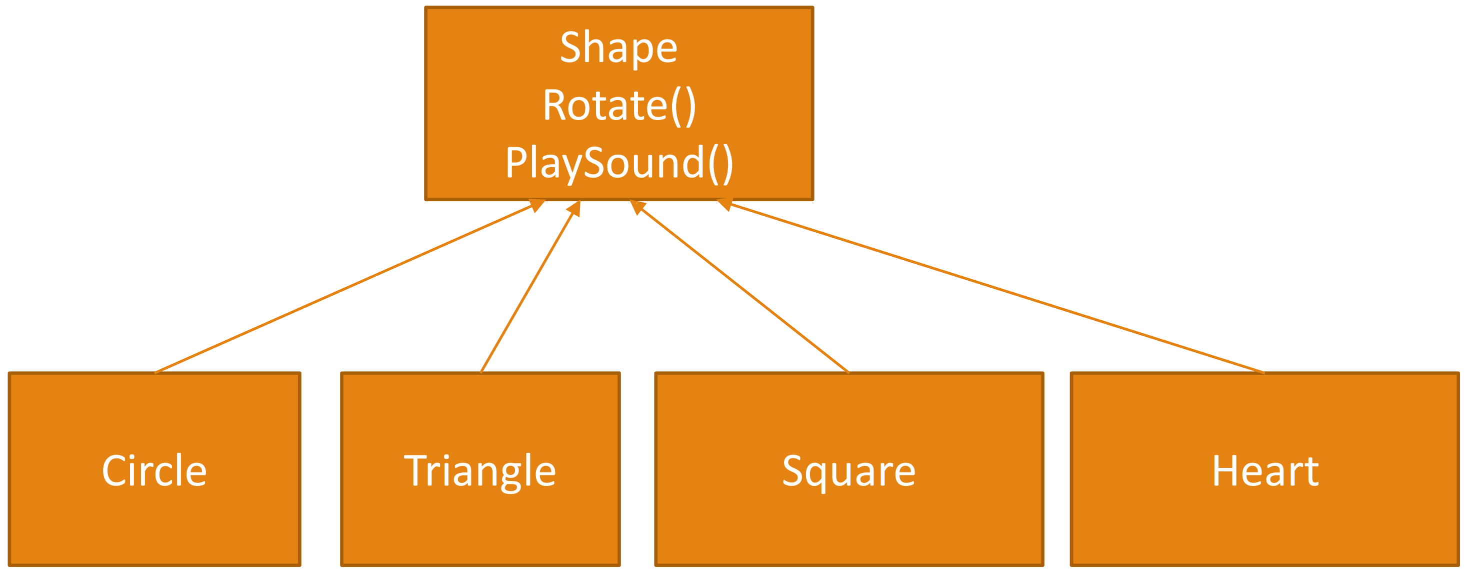
```
Class Heart
Rotate()
{
//make the shape rotate 360 degrees
}
PlaySound( )
{
//play the .mp3 file
}
```

What the classes have in common??

Object Oriented Programming

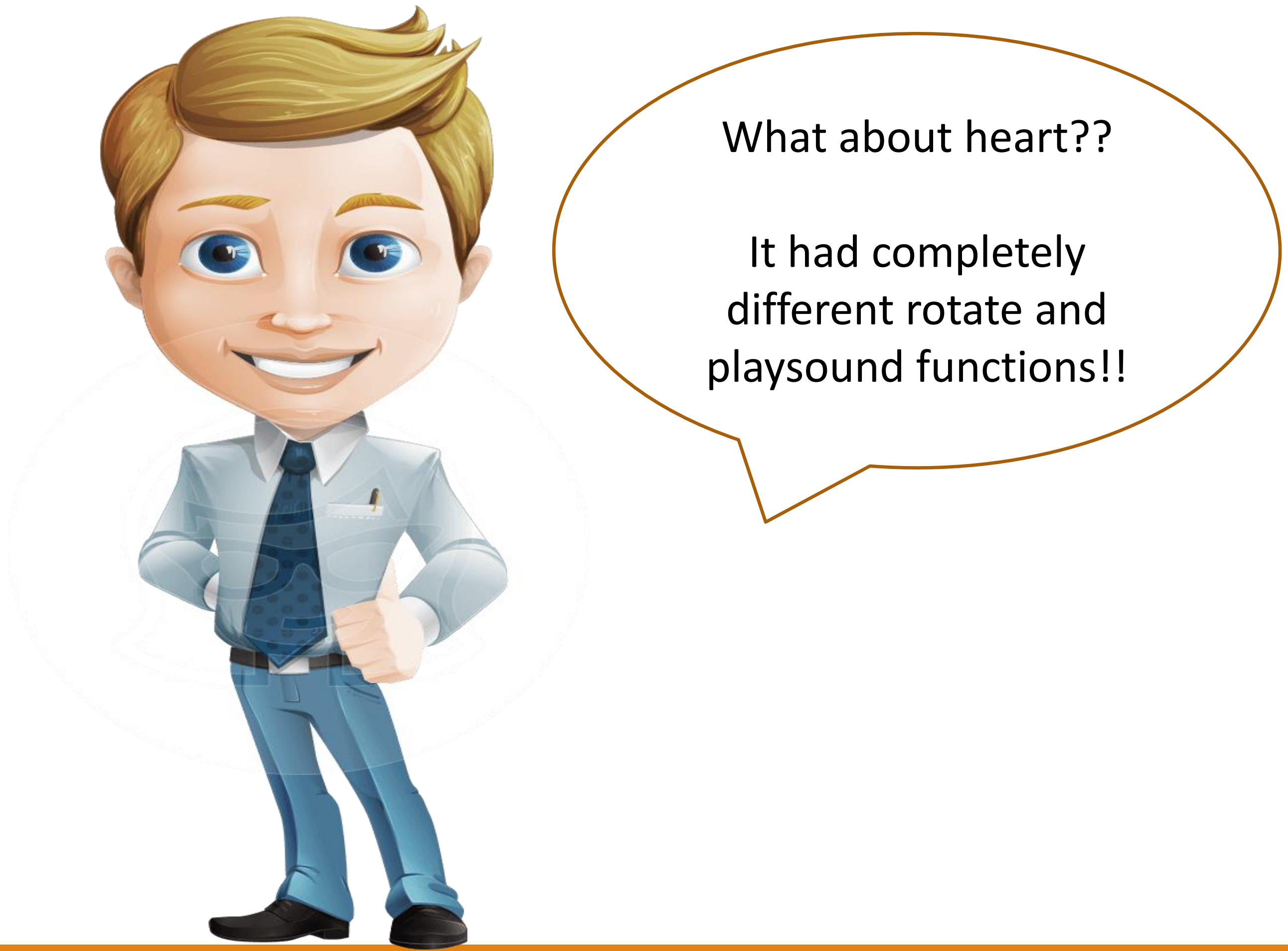
DESIGN the Base CLASS: They all are shapes and they rotates and play sound

```
Class Shape
Rotate()
{
//common features
}
//PlaySound( )
{
//common features
}
```



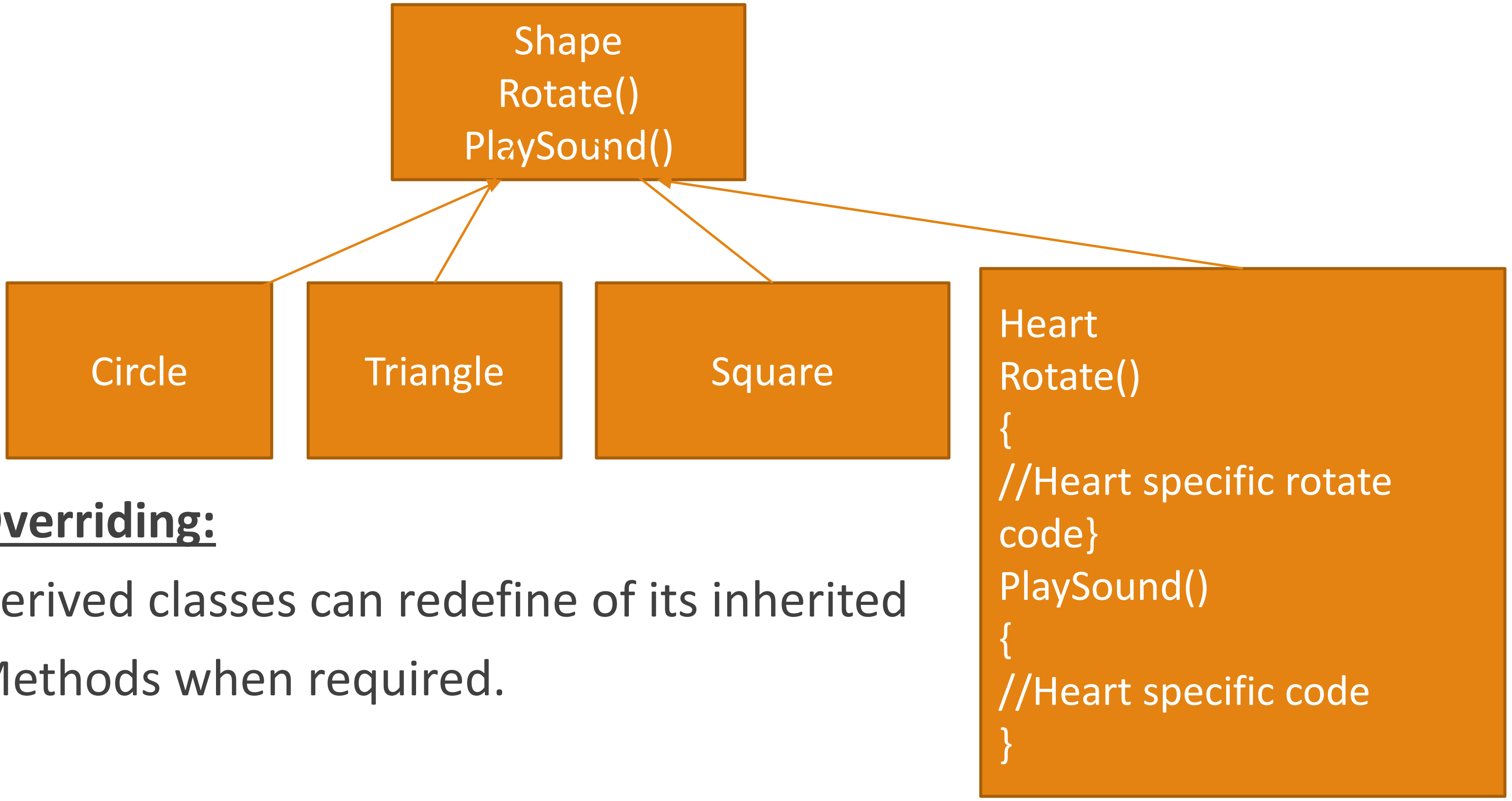
Inheritance: Base and Derived class Relationship

Object Oriented Programming


What about heart??

It had completely different rotate and playsound functions!!

Object Oriented Programming



Heart
Rotate()
{
//Heart specific rotate code}
PlaySound()
{
//Heart specific code
}

<u>Overriding:</u>

Derived classes can redefine of its inherited Methods when required.

Origin of C++

- The first program was created by toggling switches on the front panel of the computer.
  - Suitable for small programs- **machine language**
- Next, assembly language was invented. – **mid level language**
- In 1950s, first **high level language**- FORTRAN was invented.
  - Several thousand lines long program can be written
- In 1960, structure programming language was invented. – Algol and Pascal
  - Even it fails after a certain size.
- To allow more complex program, OOP was invented.

Origin of C++

- C++ extensions of C were first invented by Bjarne Stroustrup in 1979 and it was called “C with classes”- His goal was to add object-oriented programming into the C language for his Ph.D thesis.
- became C++ in ’83.
- First revision 1985
- Second revision 1990
- Third revision 1994 (ISO standardization)

Features:

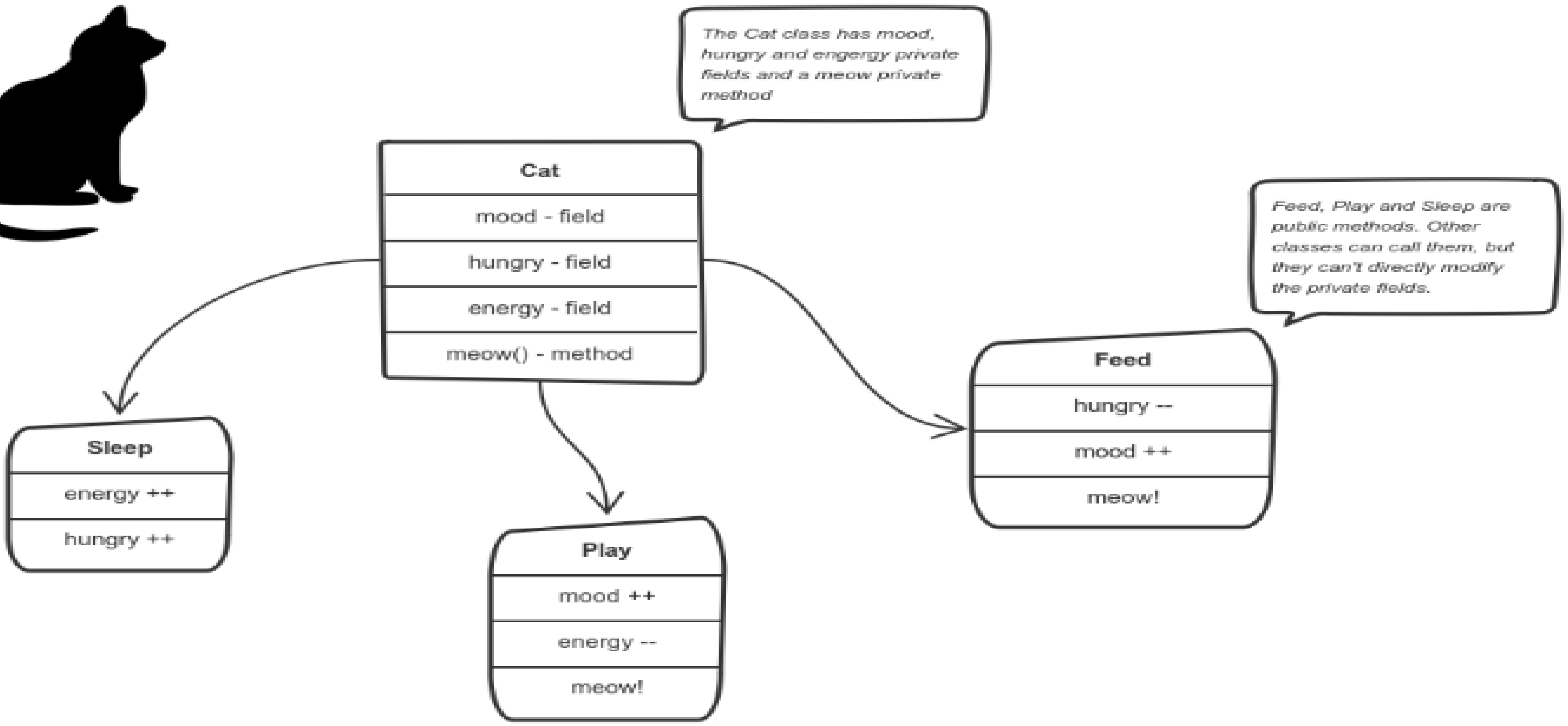
- ❑ Encapsulation
- ❑ Polymorphism
- ❑ Inheritance

Encapsulation

- ❑Encapsulation is achieved when each object keeps its state **private**, inside a class.
- ❑Other objects don’t have direct access to this state.
- ❑Instead, they can only call a list of public functions — called methods.

- ❑Let’s say we’re building a tiny Sims game. There are people and there is a cat. They communicate with each other. We want to apply encapsulation, so we encapsulate all “cat” logic into a CAT class.

Encapsulation


The Cat class has mood, hungry and engergy private fields and a meow private method
Cat
mood - field
hungry - field
energy - field
meow() - method
Feed, Play and Sleep are public methods. Other classes can call them, but they can't directly modify the private fields.
Sleep
energy ++
hungry ++
Play
mood ++
energy --
meow!
Feed
hungry --
mood ++
meow!

Encapsulation

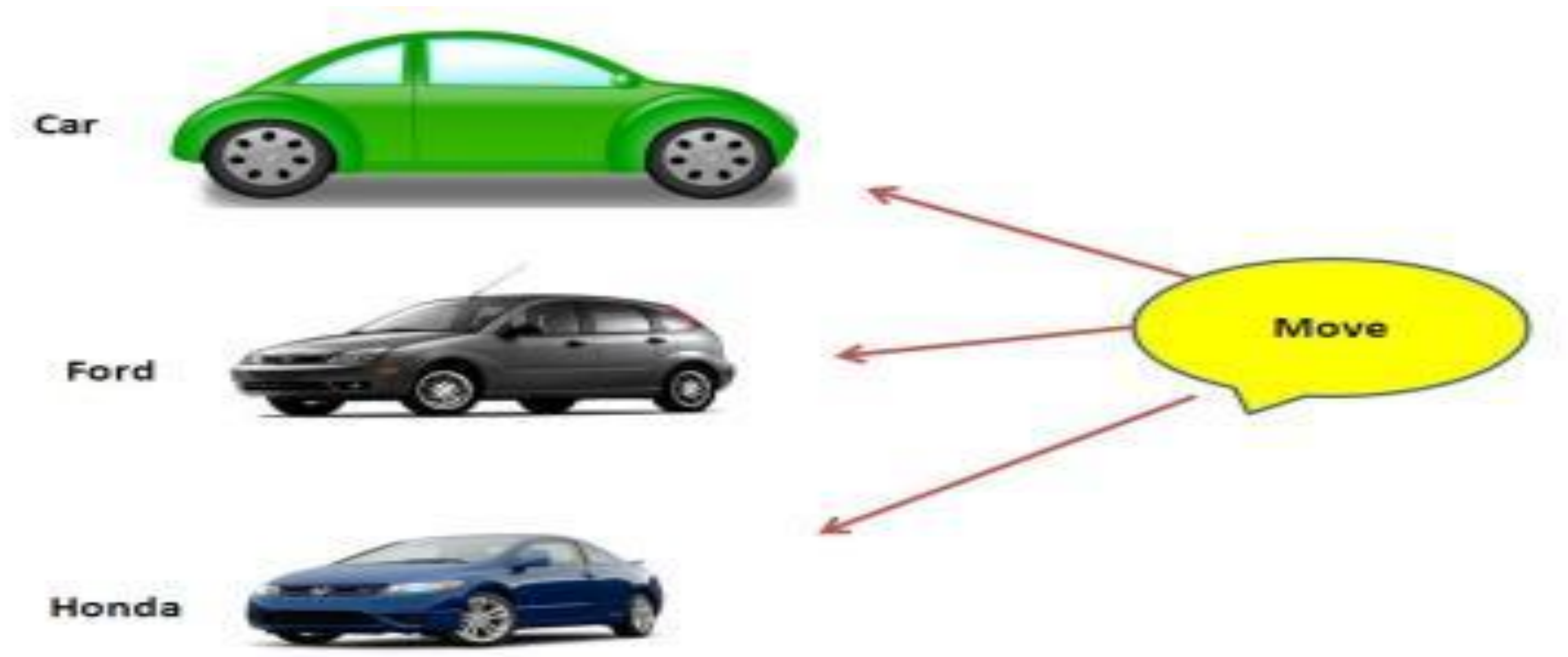
- Here the “state” of the cat is the **private variables**
  - Mood
  - Hungry
  - Energy
  - It also has a private method meow(). It can call whenever it wants, other classes can not tell the cat when to call.
  - What they can do is defined in the **public methods**
  - Sleep
  - Play
  - Feed
  - Each of them modifies the internal state somehow and may invoke meow().
  - Thus, the binding between the private state and public methods is made.

This is encapsulation.

Polymorphism

- Polymorphism means “many shapes” in Greek.
- One interface, multiple methods
- It allows one name to specify a general class of actions.
- Within a general class of actions, the specific action to be applied is determined by the type of data.

Polymorphism

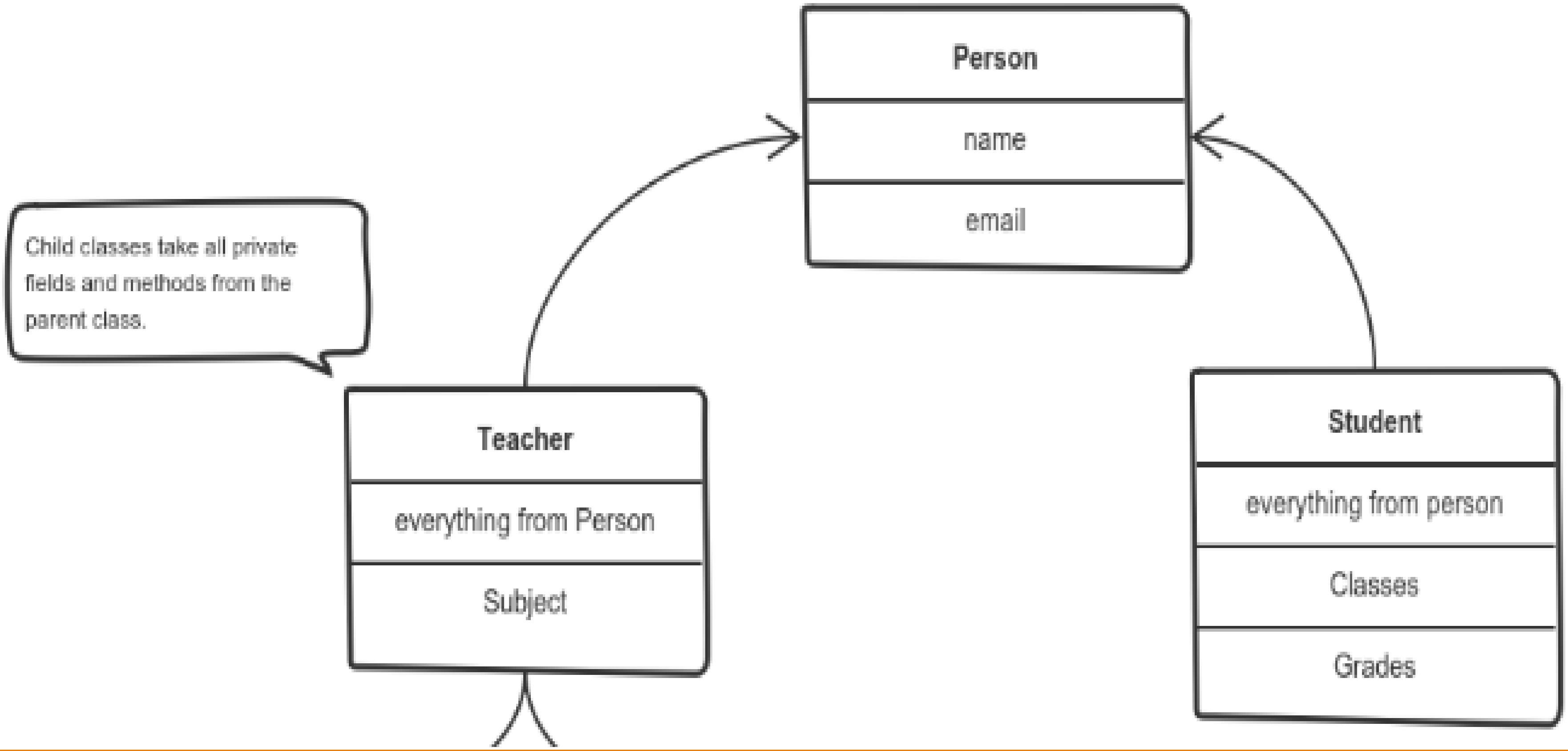


- Car uses normal engine to move
- Ford uses V engine to move
- Honda uses i-vtec technology to move

Inheritance

- We saw how encapsulation and polymorphism can help us develop and maintain a big codebase.
- But do you know what is another **common problem** in OOP design?
  - Objects are often very similar. They share common logic. But they’re not **entirely** the same.
- So, how do we reuse the common logic and extract the unique logic into a separate class?
  - One way to achieve this is **inheritance**.
- It means that you create a (child) class by deriving from another (parent) class. This way, we form a hierarchy.
- The child class reuses all fields and methods of the parent class (common part) and can implement its own (unique part).

Inheritance



C++ Console

- ❑ cin>> instead of scanf()
- ❑cout<< instead of printf()

```
int main()
{
 int i, j, k;
cin>>i;
cin>>j>>k;
cout<<i<<“ ”<<j<<endl;
cout<<k;
}
```

C++ Console

```
int main()
{
 char s[40];
cout<<“Enter Input:”<<endl;
cin>>s;
cout<<“Output”<<endl;
cout<<s;
}
```

```
Enter Input:
Hello World
Output:
Hello
```

C++ Console

```
int main()
{
char s[40];
cout<<“Enter Input:”<<endl;
cin.get(s, 40);
cout<<“Output”<<endl;
cout<<s;
}
```

```
Enter Input:
Hello World
Output:
Hello World
```

Classes

- Class is a mechanism that is used to create objects

```
class class-name
{
//private functions and variables
public:
//public functions and variables
};
```

- Functions and variables declared inside a class declaration are said to be the members of the class.

Access Modifiers in C++

- ❑ Access Specifiers/Modifiers in a class are used to set the accessibility of the class members.
- ❑ That is, it sets some restrictions on the class members not to get directly accessed by the outside functions.
- ❑ There are 3 types of access modifiers available in C++:
  - ❑ **Public-** The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.
  - ❑ **Private-** Only the member functions or the friend functions(will learn later) are allowed to access the private data members of a class.
  - ❑ **Protected-** We will learn about the protected members later.

More on class

Class can be considered like other data types

So we can

- Assign objects
- Create Array of objects
- Take pointer/references of objects
- Allocate and free objects
- Pass objects to functions
- Return objects from functions
- Make an object member of another object
- Use sizeof operator with class

Class

```
class myclass
{
 int a;
 public:
 void seta(int x);
 int geta();
};
```

- Integer a is a private member: so cannot be accessed from outside of the class
- seta() and geta() both are public functions.
- To define a member function, we must link the name of the class with the name of the function

```
ret-type class-name :: func-name (parameter list)
{  //body of the function
}
```

Scope resolution operator

Class

```
void myclass::seta(int x)
{
   a=x;
}
int myclass::geta()
{
   return a;
}
```

Class

- ❑ The declaration of myclass did not define any objects of type myclass. – it only defines the type of object that will be created when one is actually declared.
- ❑To declare an object:

myclass ob1, ob2;

- ❑ An object occupies memory space, but a type definition does not.
- ❑Each object of a class has its own copy of every variable declared within the class.
- ❑Once an object is declared, it can reference public member using dot(.)

```
ob1.seta(10);    //sets ob1's version of a to 10
ob2.seta(100); //sets ob2's version of a to 100
```

Put them all together..

```
#include<iostream>
using namespace std;
class myclass
{
   int a;
public:
   void seta(int x);
   int geta();
};

void myclass::seta(int x)
{
   a=x;
}
int myclass::geta()
{
   return a;
}
```

```
int main()
{
myclass ob1, ob2;
ob1.seta(10);
ob2.seta(100);
cout<<ob1.geta()<<endl;
cout<<ob2.geta()<<endl;
}
```

Output:

```
10
100
```