OOP Concepts

Object-Oriented Programming

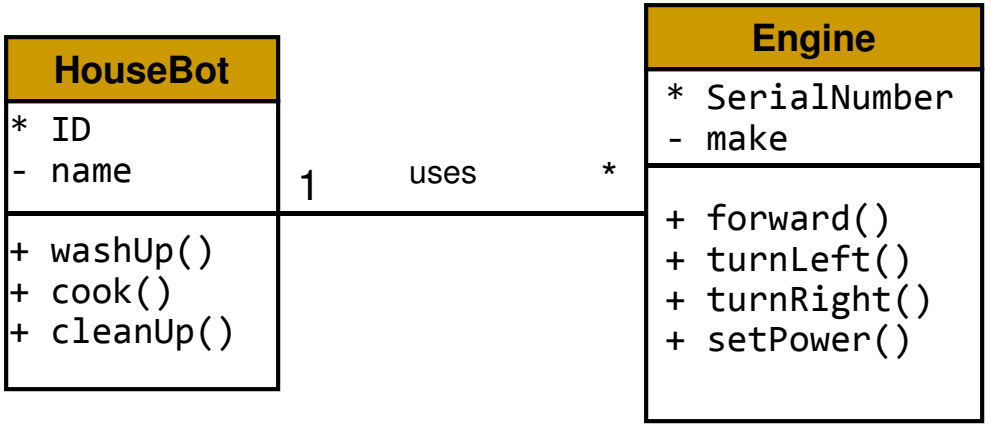
Outline

- What is object-oriented programming?
- Procedural vs. object-oriented programming
- OOP concepts

Readings:

- HFJ: Ch.2.
- GT: Ch.1.

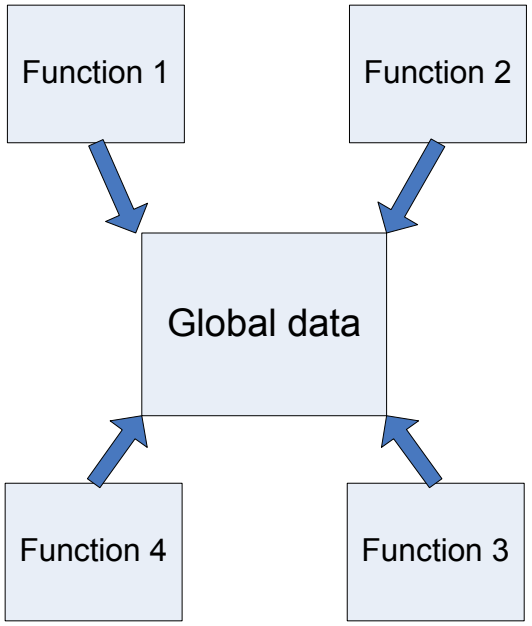
What is OOP?



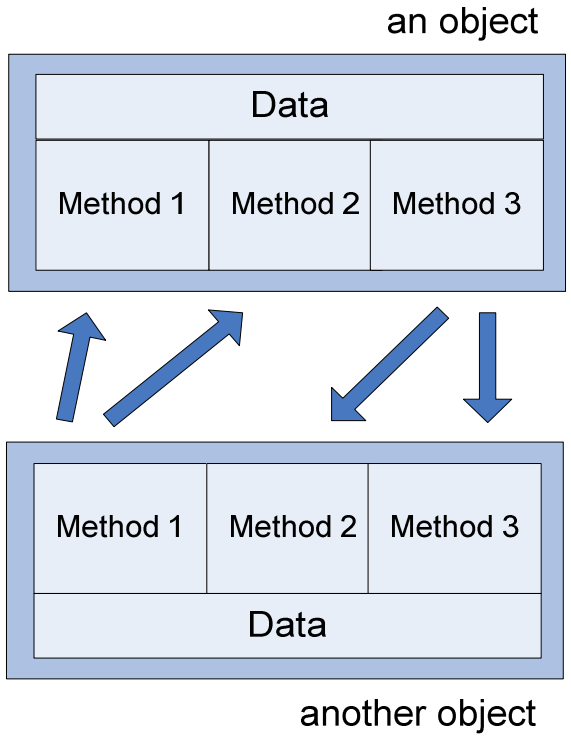
- OOP
  - Map your problem in the real world
  - Define “things” (objects) which can either do something or have something done to them
  - Create a “type” (class) for these objects so that you don’t have to redo all the work in defining an objects properties and behavior
- An OO program: “*a bunch of objects telling each other what to do by sending messages*”. (Smalltalk)

Procedural vs. Object-oriented

- Procedural program
  - passive data



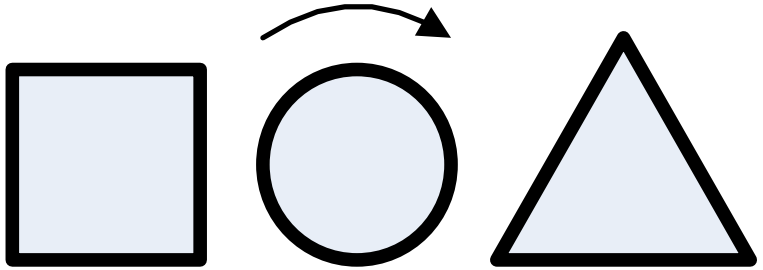
- Object-oriented program
  - active data



Procedural vs. Object-oriented Example

- Given a specification:

There will be shapes on a GUI, a square, a circle, and a triangle. When the user clicks on a shape, the shape will rotate clockwise 360o, (i.e. all the way around) and play an AIF sound file specific to that particular shape.



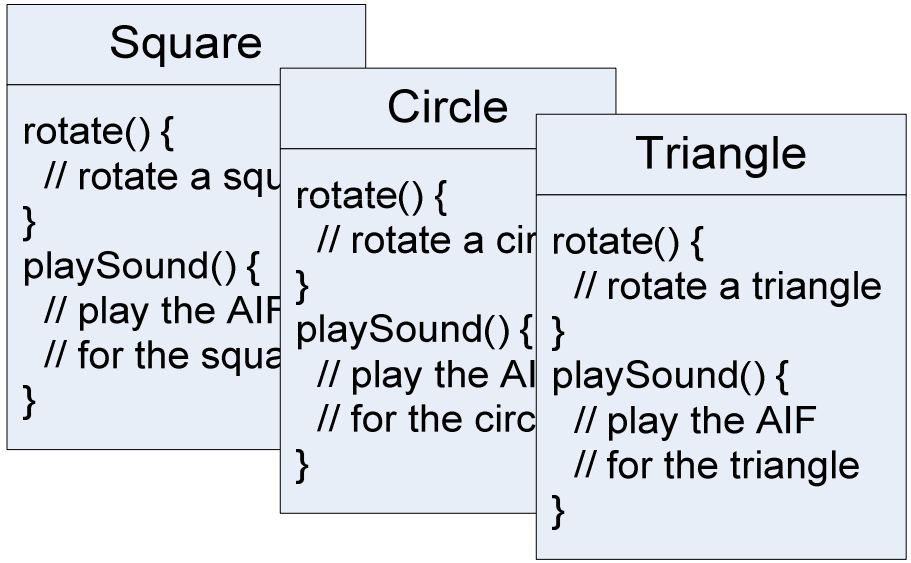
- Procedural solution?
- Object-oriented solution?

Procedural vs. Object-oriented Example

- Procedural

```
rotate(shapeNum) {
  //make the shape
  //...rotate 360o
}
playSound(shapeNum) {
  //use shapeNum to look up
  //...which AIF to play
  //and play it
}
```

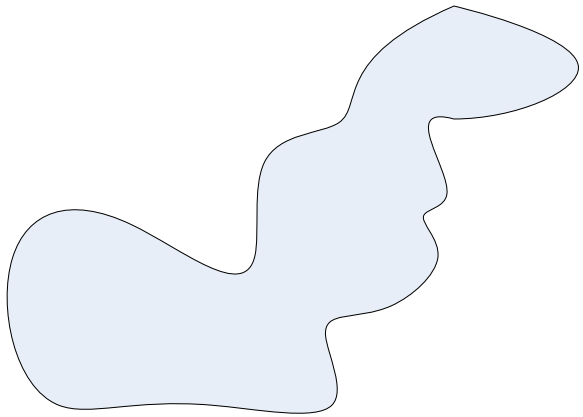
- Object-oriented



Procedural vs. Object-oriented Example

Then comes a change to the specification:

There will be an amoeba shape on the screen, with the others. When the user clicks on the amoeba, it will rotate like the others, and play a .hif sound file.



- Procedural solution?
- Object-oriented solution?

Procedural vs. Object-oriented Example

- Procedural

  playSound() has to change

```
playSound(shapeNum) {
  // if the shape is not amoeba
    //use shapeNum to look up
    //...which AIF to play
    //and play it
  // else
    //play amoeba .hif sound
}
```

- Object-oriented

  class Amoeba is added

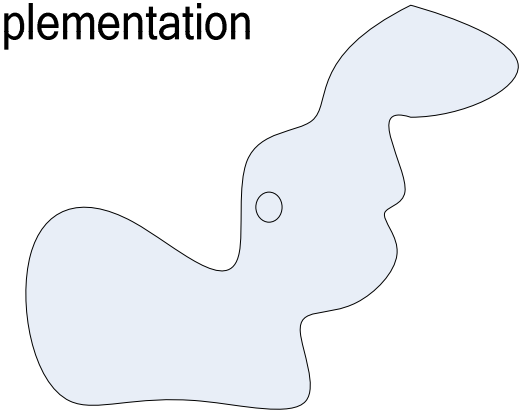
Amoeba

rotate() {
  // rotate an amoeba
}
playSound() {
  // play the .hif
  // for the the amoeba
}

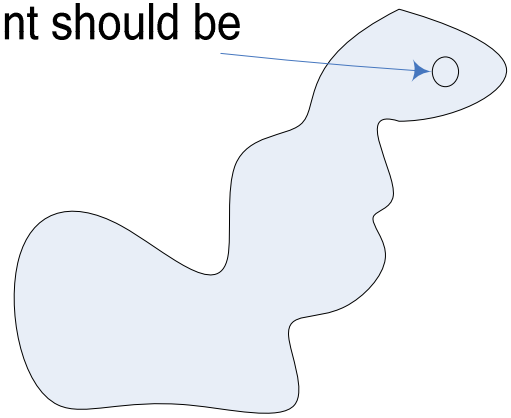
Procedural vs. Object-oriented Example

Then comes another change to the specification:

Amoeba rotate point in current implementation



Where the amoeba rotate point should be



- Procedural solution?
- Object-oriented solution?

Procedural vs. Object-oriented Example

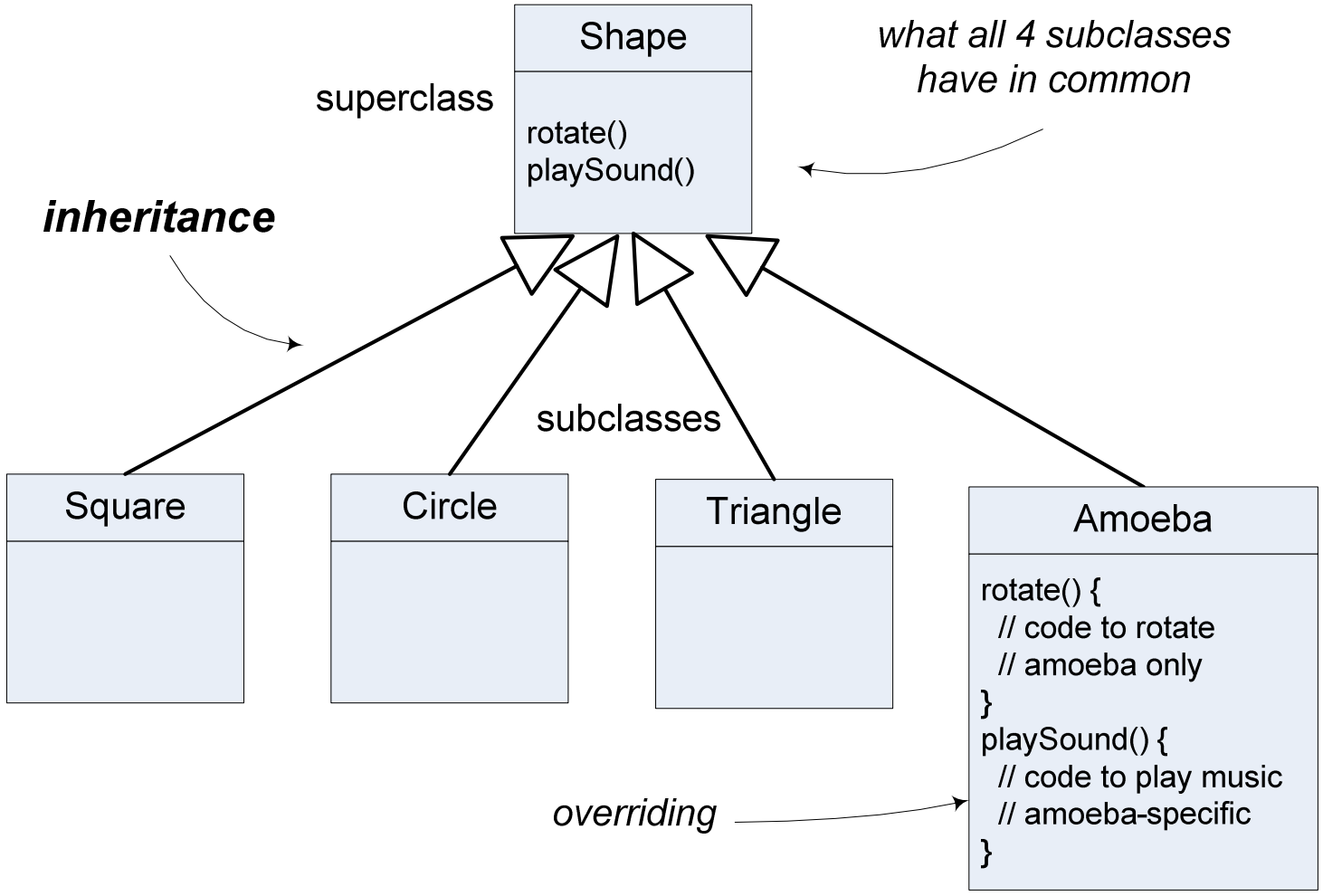
- Procedural
  - rotate() is modified
  - **so is ALL the related code**
- Object-oriented
  - class Amoeba is changed
  - the rest is NOT affected

```
rotate(shapeNum, xPt, yPt ) {
  // if the shape is not amoeba
    //calculate center point
    //based on a rectangle
    //then rotate
  // else
    //use xPt,yPt as
    //the rotation point offset
    //and then rotate
}
```

Amoeba

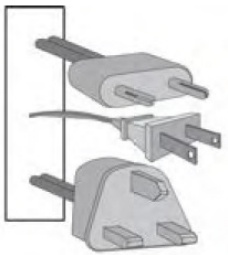
int xPoint
int yPoint
rotate() {
  // rotate an amoeba
  // using xPoint, yPoint
}
playSound() {
  // play the .hif
  // for the the amoeba
}

OOP solution



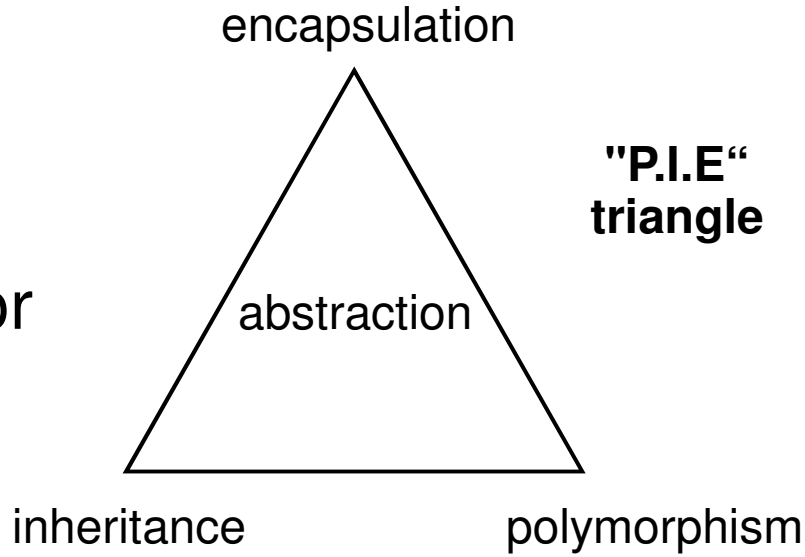
The goals of object-oriented design

- **Robustness**: software is capable of handling unexpected inputs that are not explicitly defined for its application.
  - Nuclear plant control software
  - Airplane control software
- **Adaptability**: software that can evolve over time in response to changing conditions in its environment.
  - Web browsers and Internet search engines typically involve large programs that are used for many years.
- **Reusability**: the same code should be usable as a component of different systems in various applications.
  - Save time and money



Important OO concepts

- Abstraction
- Objects & Class
  - Object state and behavior
  - Object identity
  - Messages
- Encapsulation
  - Information/implementation hiding
- Inheritance
- Polymorphism



"P.I.E" triangle

Abstraction

- Abstraction: to distill a complicated system down to its most fundamental parts and describe these parts in a simple, precise language.
  - naming the parts
  - explaining their functionality
- Examples:
  - Design of data ➔ abstract data types (ADT)

Abstraction

Sue's car:
Fuel: 20 liter
Speed: 0 km/h
License plate: "143 WJT"

Martin's car:
Fuel: 49.2 liter
Speed: 76 km/h
License plate: "947 JST"

Tom's car:
Fuel: 12 liter
Speed: 40 km/h
License plate: "241 NGO"

Automobile:

- fuel
- speed
- license plate

- speed up
- slow down
- stop

Objects

Objectname: suesCar

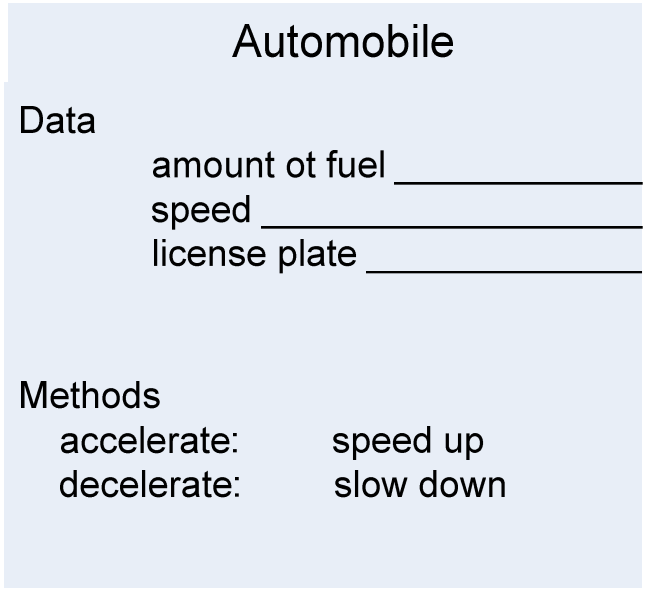
amount of fuel: 20 lit
speed: 0 km/h
license plate: “143 WJT”

Objectname: martinsCar

amount of fuel: 49.2 lit
speed: 76 km/h
license plate: “947 JTS”

Objectname: tomsCar

amount of fuel: 12 lit
speed: 40 km/h
license plate: “241 NGO”



- An object has
  - State
    - Changes over time
  - Behavior
    - What the object does in response to messages
  - Identity
    - What makes the object unique

State

- Given by object's attributes

Dave
Age: 32
Height: 6' 2"

Brett
Age: 35
Height: 5' 10"

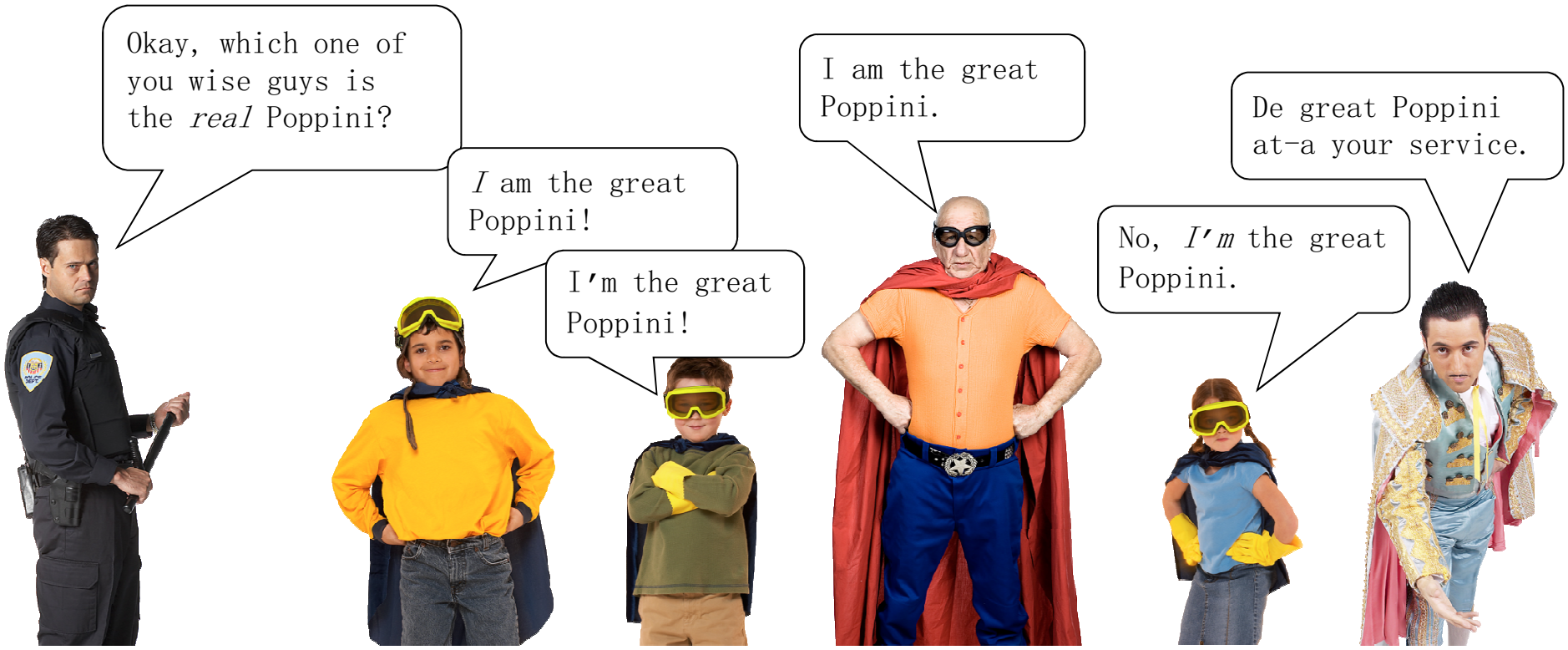
Gary
Age: 61
Height: 5' 8"

Behavior

- What the object can do responding to a message.

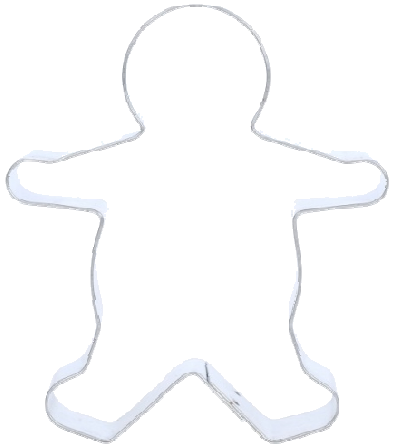
Identity

- Something to distinguish between objects.

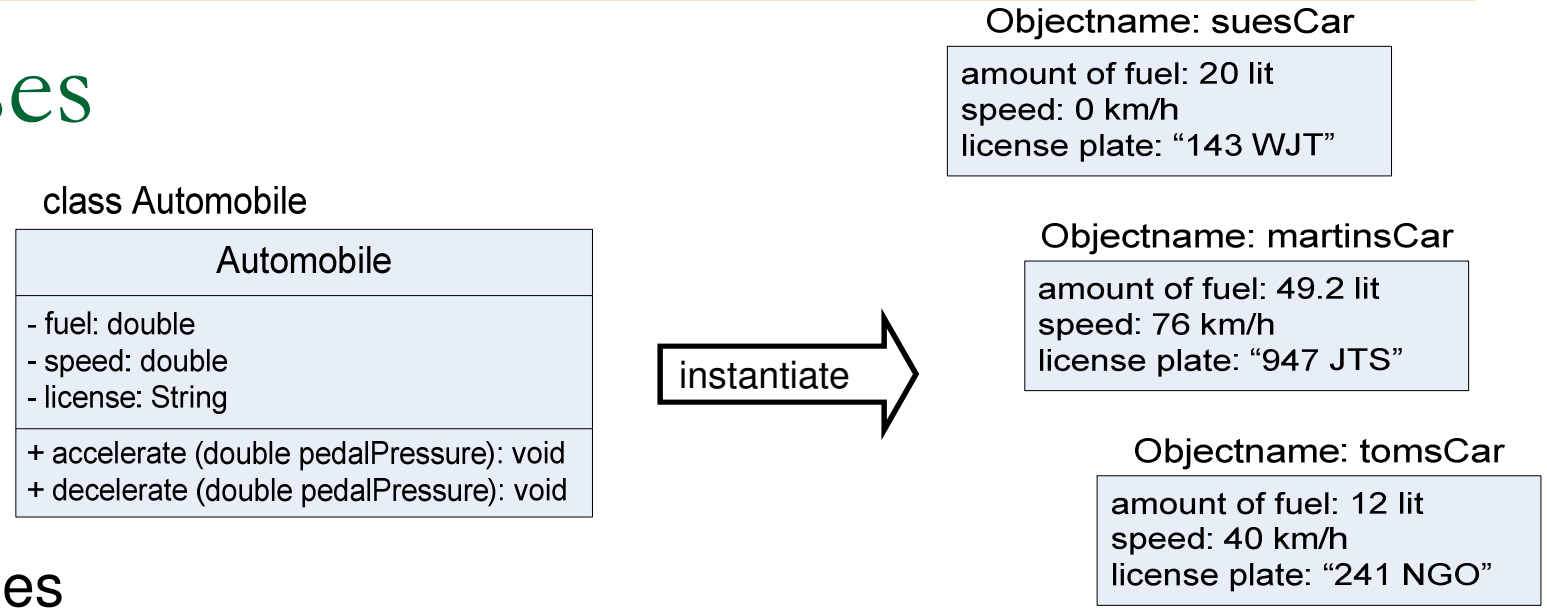


Classes

- Define the properties and behavior of objects
- Can have behavior and properties that are defined in the class but are independent of the individual objects

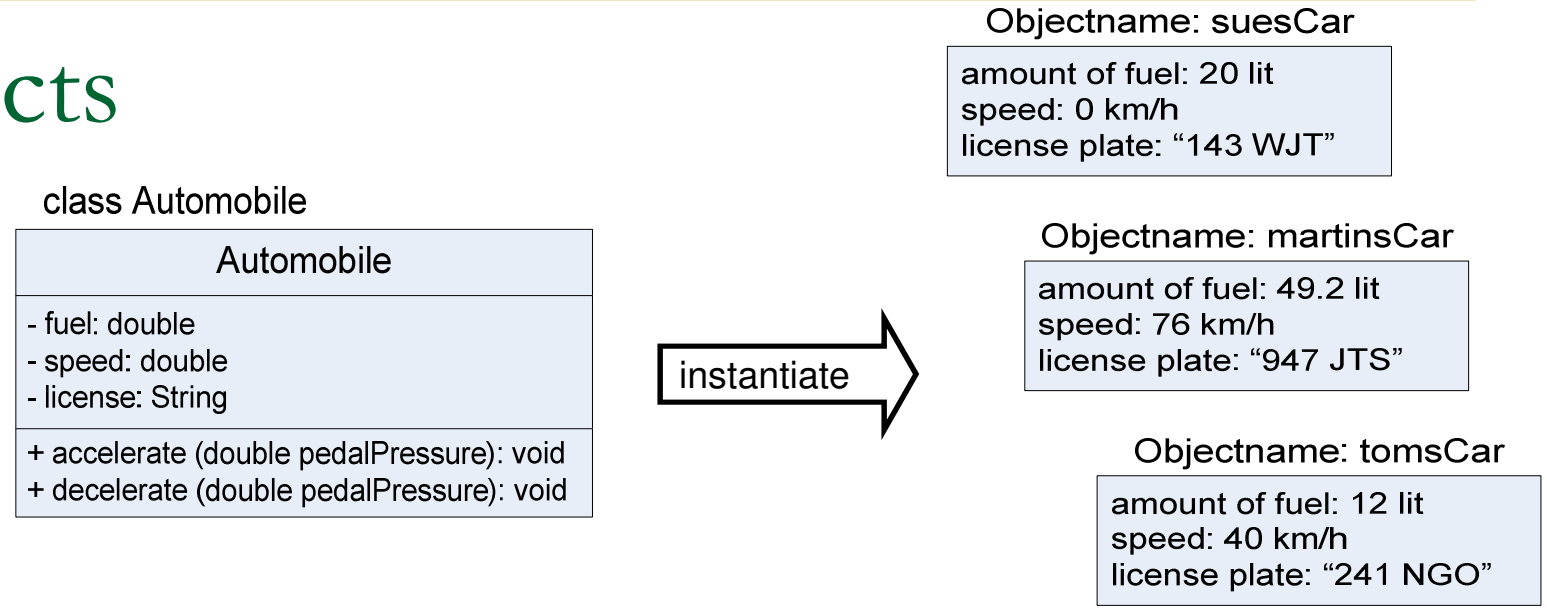


Classes



- Classes
  - are the templates to create objects (instantiate).
  - Each object has the same structure and behaviour as the class from which it was created
- “Data type – Variable” relation
  - Classes are what we design and code. Class definitions make up programs.
  - Objects are what are created (from a class) at run-time

Objects



- State → Attributes / Instant variables
  - Variables holding state information of the object
- Behavior → Methods
  - Operations/services performed on the object.

Messages


```
myCar.accelerate(80);
```

- A means for object A to request object B to perform one method of B's.
- A message consists of:
  - Handle of the destination object – host (`myCar`)
  - Name of the method to perform (`accelerate`)
  - Other necessary information – arguments (`80`)
- In effect, a message is a function call with the host object as the implicit argument (method invocation)
- However, the concept of messages has great significance to OOP:

Data become active!

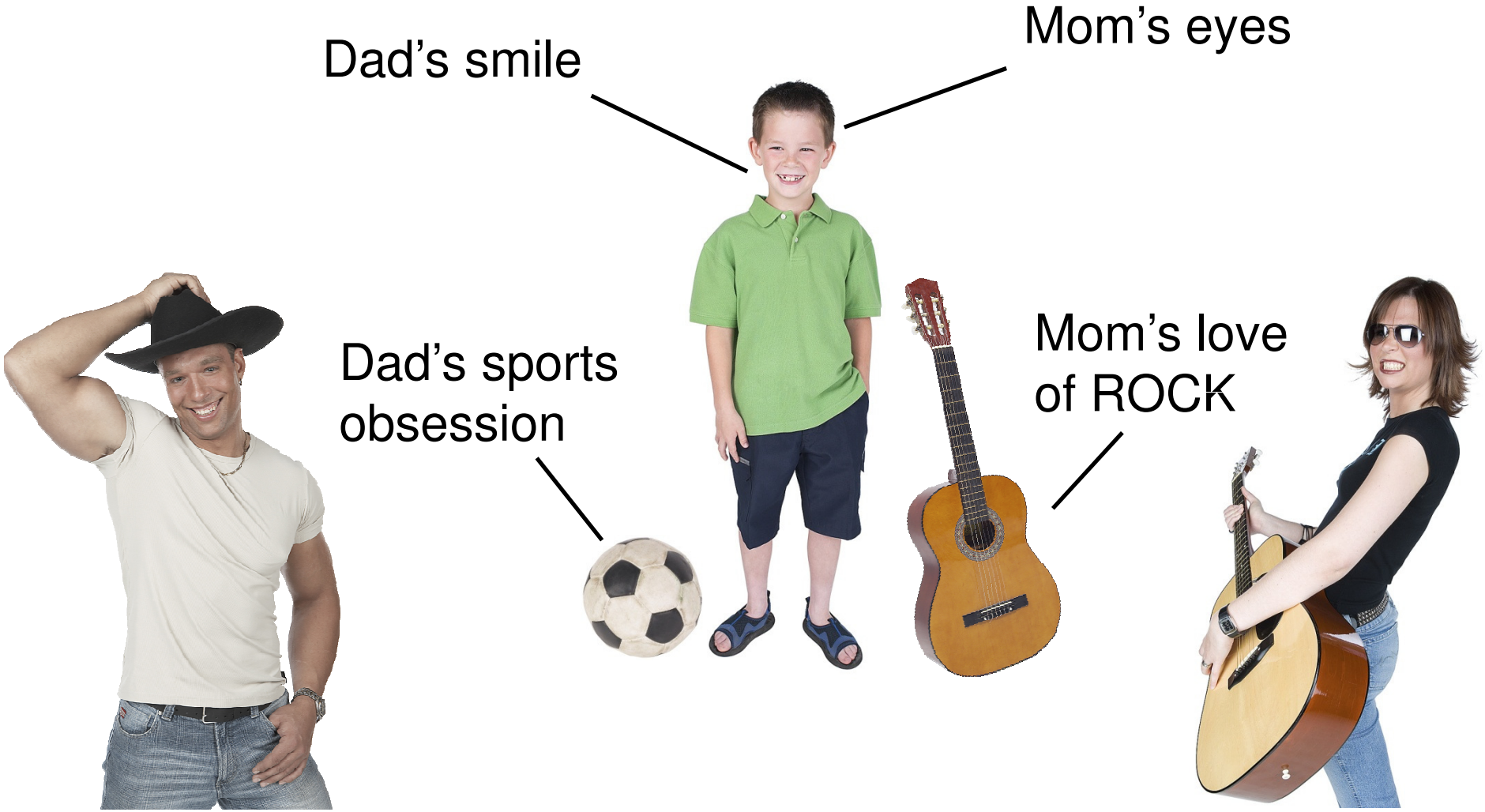
Encapsulation

Encapsulation / Information hiding

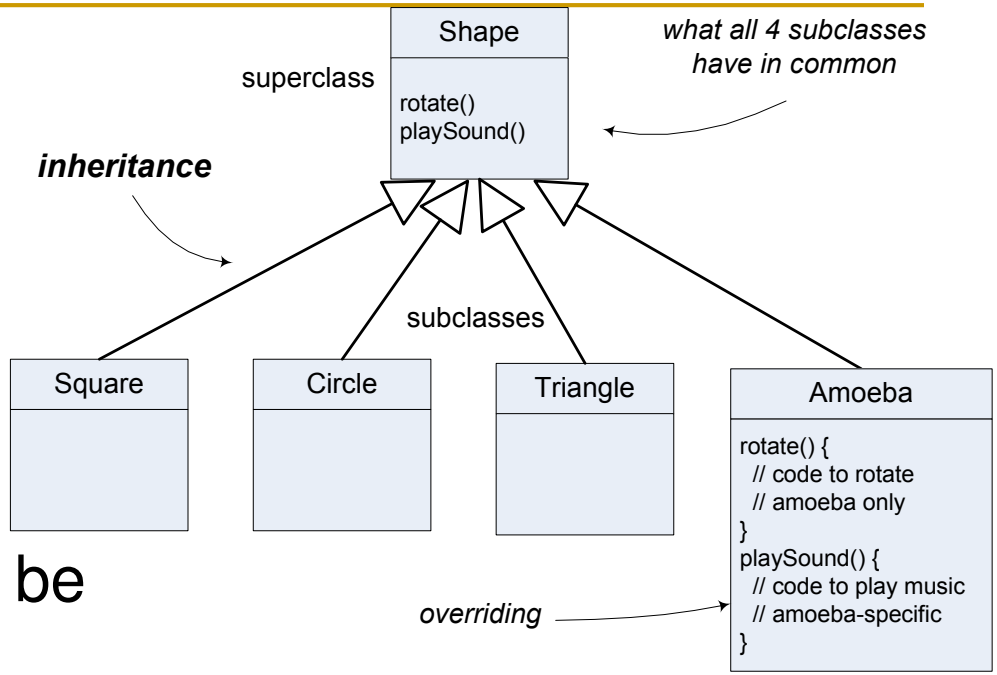
```
class Car {
  public void setSpeed(…) {
    // check validity
    // set new values
    ...
  }
  ...
  private double speed;
}
```

- **Encapsulation**:
  to group related things together
  - Functions/procedures encapsulate instructions
  - Objects encapsulate data and related procedures
- **Information hiding**: encapsulate to hide internal implementation details from outsiders
  - Outsiders see only interfaces
  - Programmers have the freedom in implementing the details of a system.
  - Hence, the ability to make changes to an object's implementation without affecting other parts of the program

Inheritance



Inheritance



- "is-a" relations
- The general classes can be specialized to more specific classes
- Reuse of interfaces & implementation
- Mechanism to allow derived classes to possess attributes and operations of base class, as if they were defined at the derived class
- We can design generic services before specialising them

Polymophism

Polymorphism:

- "more than one form"

Object polymorphism:

- Different types of objects can respond to the ***same message***. And they can respond ***differently***.
- Example: the square and the amoeba both can receive message *rotate()*, they respond by doing different things.

OOP languages

- Some OOP features can be implemented in C or other procedural programming languages, but not enforced by these languages
- OOP languages: OOP concepts are embeded in and enforced by the languages.
- OOP languages vary in degrees of object-oriented
  - Pure: Smalltalk, Eiffel, Ruby, JADE..
  - Original OO plus some procedural features: Python, Java (very high), C++ (mixed), C#..
  - OO features as extension: VB.NET, Fortran 2003, PHP, Perl..

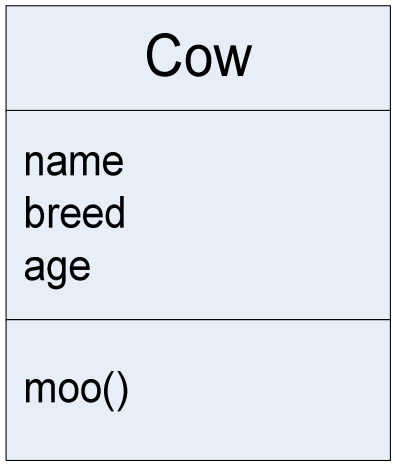
Example

```
class Cow {
  String name;
  String breed;
  int age;

  void moo() {
    System.out.println("Moo...!");
  }
}
```

instance variables

a method




```
public class CowTestDrive {
  public static void main (String[] args) {
    Cow c = new Cow();  // make a Cow object
    c.age = 2; // set the age of the Cow
    c.moo();   // call its moo() method
  }
}
```