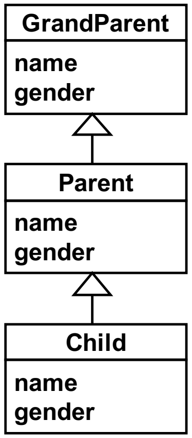
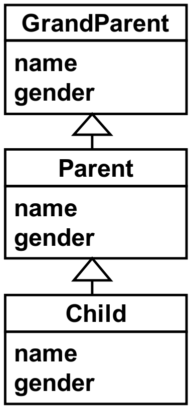
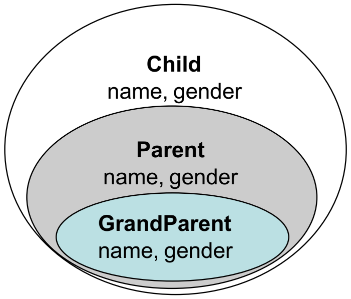
Recursive Associations

A Wrong Class Design

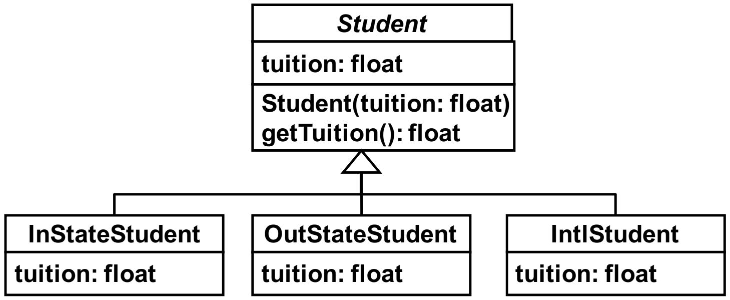

GrandParent
name
gender
Parent
name
gender
Child
name
gender

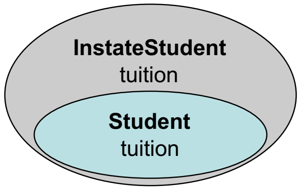
- A classical design error in/since mid ’80.
  - A parent inherits a grand parent’s variables and methods.
  - A child inherits a parent’s variables and methods.
- found in an OOP textbook published in 2009.


GrandParent
name
gender
Parent
name
gender
Child
name
gender

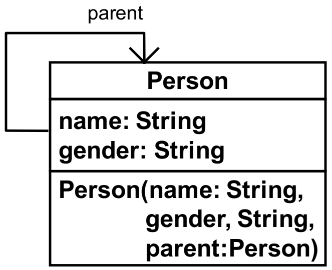

Child
name, gender
Parent
name, gender
GrandParent
name, gender

- A parent has two “name” data fields and two “gender” data fields.
- A child has three “name” data fields and three “gender” data fields.


Student
tuition: float
Student(tuition: float)
getTuition(): float
InStateStudent
tuition: float
OutStateStudent
tuition: float
IntlStudent
tuition: float


InstateStudent
tuition
Student
tuition

Right Design

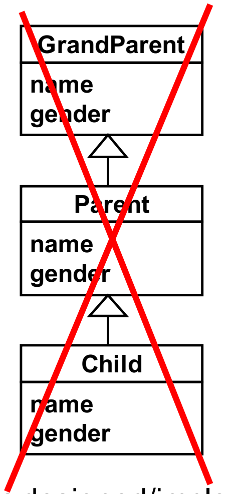


parent

Person

name: String
gender: String

Person(name: String,
gender, String,
parent:Person)



~~GrandParent~~

~~name~~
~~gender~~

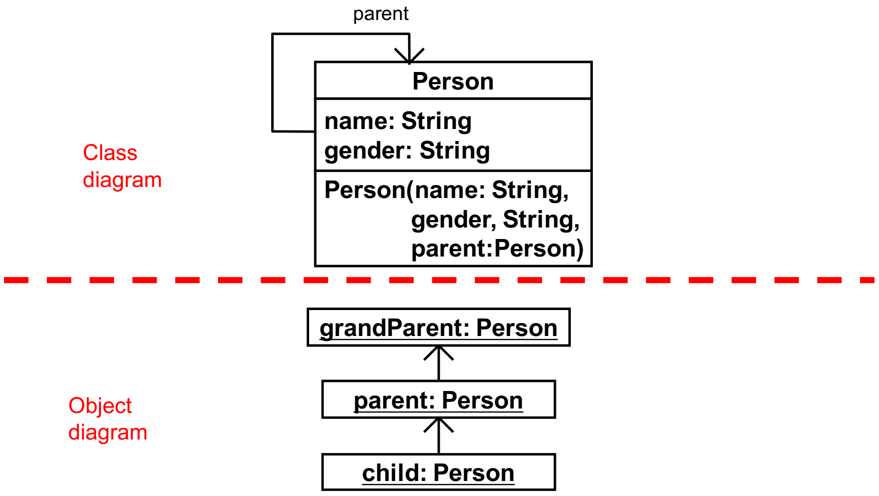
~~Parent~~

~~name~~
~~gender~~

~~Child~~

~~name~~
~~gender~~

- Association v.s. inheritance
- A conceptual hierarchy is not always designed/implemented with a class inheritance(s).



parent

Person

name: String
gender: String

Person(name: String,
gender, String,
parent:Person)

Class diagram

Object diagram

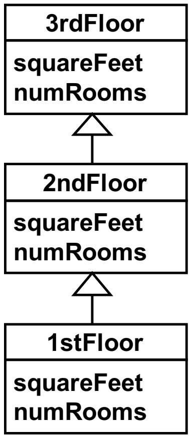
grandParent: Person

parent: Person

child: Person

- `Person grandParent = new Person(“grandpa”, “male”, null);`
- `Person parent = new Person(“dad”,”male”, grandParent);`
- `Person child = new Person(“me”, “male”, parent);`

Another Example



3rdFloor

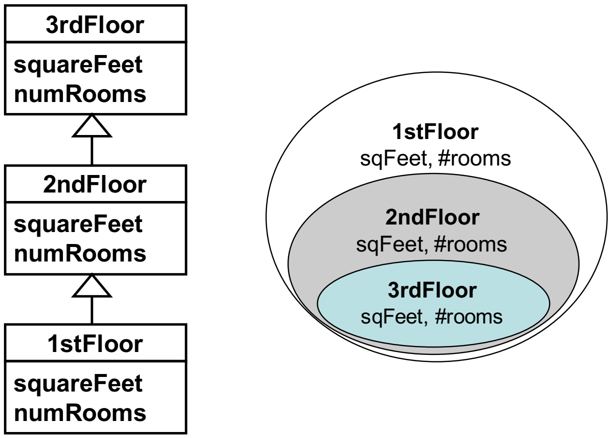
squareFeet
numRooms

2ndFloor

squareFeet
numRooms

1stFloor

squareFeet
numRooms



3rdFloor

squareFeet
numRooms

2ndFloor

squareFeet
numRooms

1stFloor

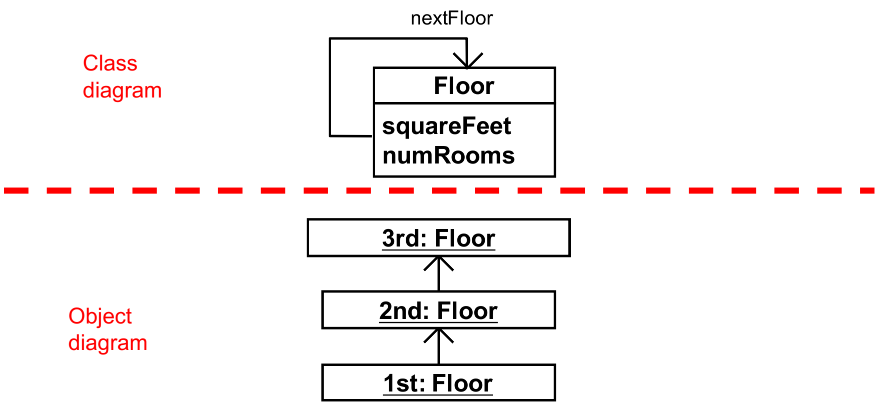
squareFeet
numRooms

1stFloor
sqFeet, #rooms

2ndFloor
sqFeet, #rooms

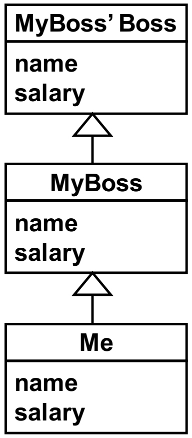
3rdFloor
sqFeet, #rooms

A conceptual hierarchy is not always designed/implemented with a class inheritance(s).

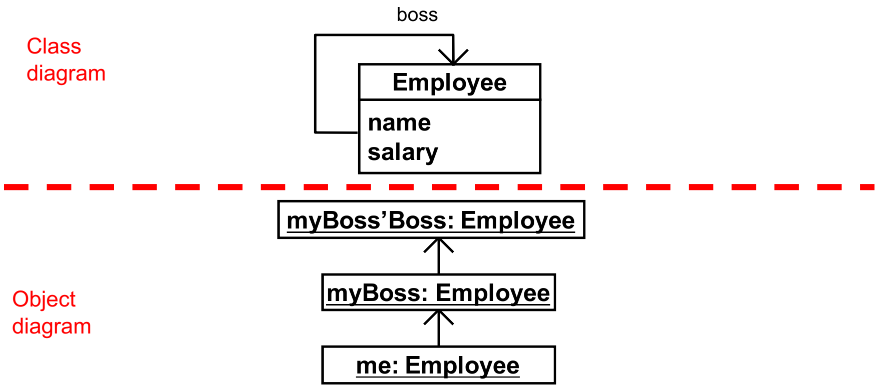


```
• Floor 3rd = new Floor(500, 2, null);
• Floor 2nd = new Floor(700, 3, 3rd);
• Floor1st = new Floor(1000, 5, 2nd);
```

One More



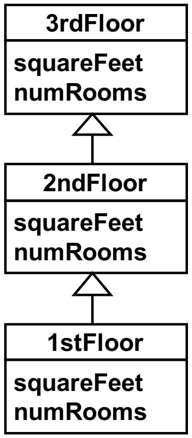
A conceptual hierarchy is not always designed/implemented with a class inheritance(s)

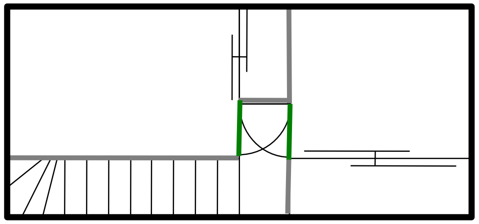


```
• Employee bossboss = new Employee(“bossboss”, 100, null);
  Employee boss= new Employee(“boss”, 70, bossboss);
  Employee me= new Employee(“me”, 50, boss);
```

- Class-instance relationships are very important.
  - The first important step to good OOD.
- If you don't perfectly understand what I am talking about…
  - write code for example classes (wrong and right ones)
  - play with them using some test code
  - This is not HW, but you will struggle with future HWs if you don't perfectly understand class-instance relationships.

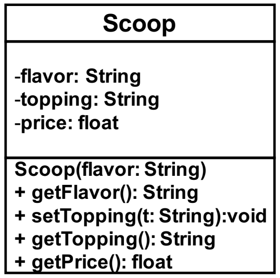
This Wrong Design Reminds Me of…





Modeling Exercise: Modeling Ice Creams

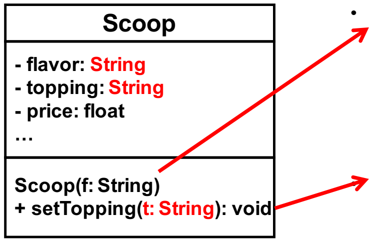
Modeling Ice Creams



```
Scoop scoop = new Scoop("vanilla");
scoop.setToopping("chocolate");
```

Most likely, you don't need setter methods for flavor and price. Be preventive!

Don't Do this.



```
•   if( f.equals("vanilla") ){
        // do something
    }else if ( f.equals("chocolate")
    ){
        // do something else
    }

    if( t.equals("strawberry") ){
        // do something
    }else if ( t.equals("chocolate")
    ){
        // do something else
    }
```

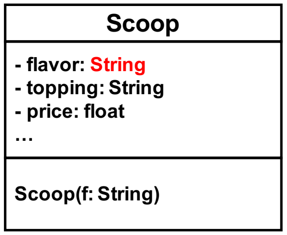
```
new Scoop("vanillaaaa");
```

An error (typo) can occur even if you carefully don't define a setter method for flavor.

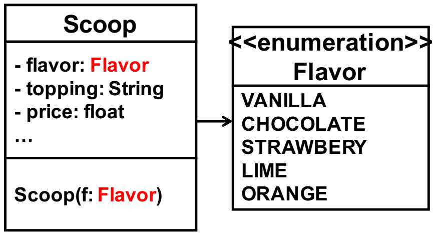
You need to write error-handling code. In the worst case, errors may not be detected at runtime.

You want to catch as many errors as possible at compile-time. Have your complier work harder!

Use an Enumeration

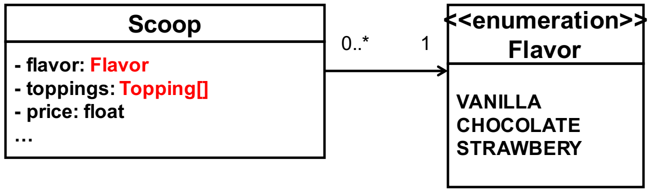


```
•  If( f.equals(“vanilla”) ){
        // do something
   }else if( f.equals(“chocolate”)
   ){
        // do something else
   }


   new Scoop(“vanilla”);
```

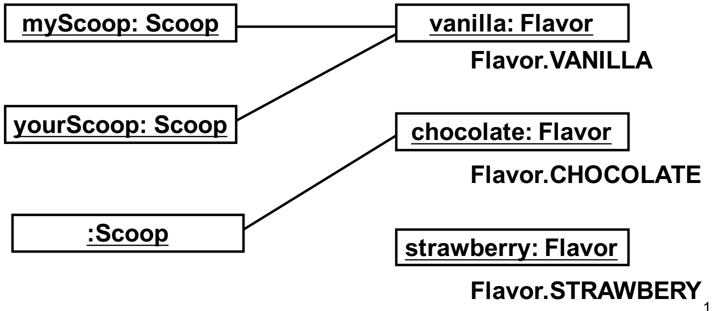


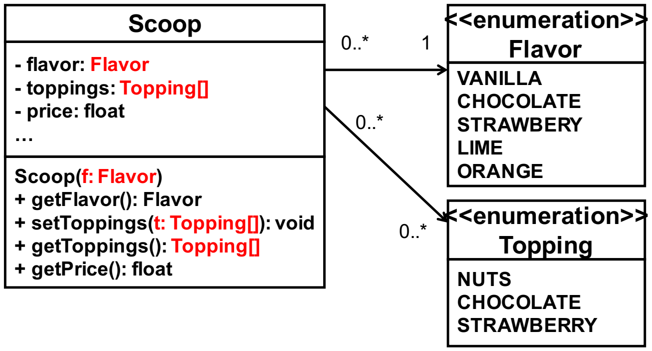
```
If( f.equals(Flavor.VANILLA) ){
   // do something
}
else if( f.equals(Flavor.CHOCOLATE)
   ){
   // do something else
}


new Scoop(Flavor.VANILLA);
```

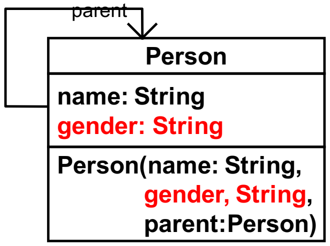
Class diagram

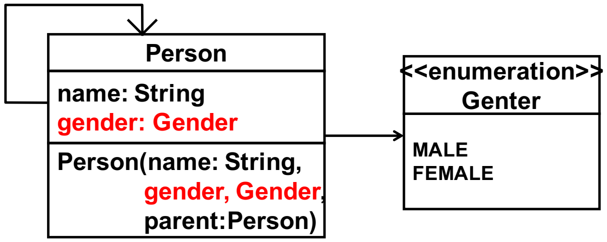


Object diagram







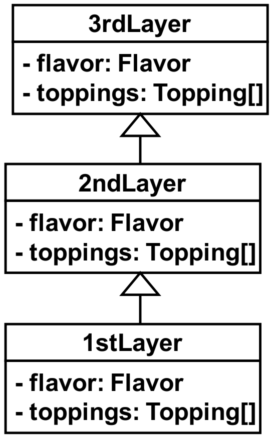


How about these ones?

On May 16, 2005, Eisenhower Junior High School set a new world record for producing the *World's Tallest Ice Cream Cone.* Three 9th Grade students produced an ice cream cone that reached a height of 13.00 inches.

http://www.graniteschools.org/jr/eisenhower/images/memorie%20pics/world_records/Tallest_Ice_Cream_Cone.jpg

Never Think This Way



3rdLayer

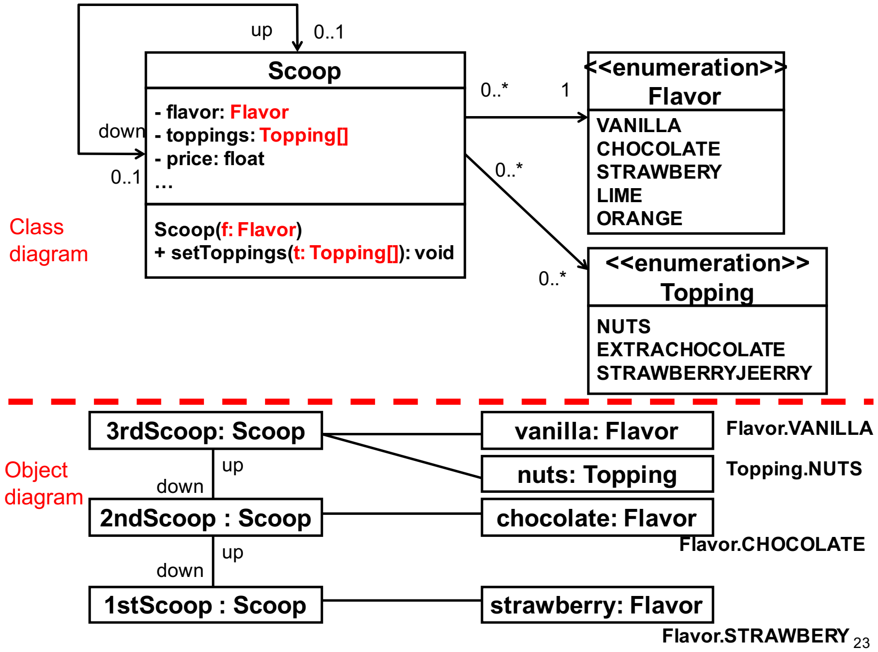
- flavor: Flavor
- toppings: Topping[]

2ndLayer

- flavor: Flavor
- toppings: Topping[]

1stLayer

- flavor: Flavor
- toppings: Topping[]



Class diagram

up 0..1

Scoop

- flavor: Flavor
- toppings: Topping[]
- price: float

...

down 0..1

Scoop(f: Flavor)

+ setToppings(t: Topping[]): void

0..* 1

<<enumeration>>
Flavor

VANILLA
CHOCOLATE
STRAWBERY
LIME
ORANGE

0..* 0..*

<<enumeration>>
Topping

NUTS
EXTRACHOCOLATE
STRAWBERRYJEERRY

Object diagram

3rdScoop: Scoop

up down

2ndScoop : Scoop

up down

1stScoop : Scoop

vanilla: Flavor Flavor.VANILLA

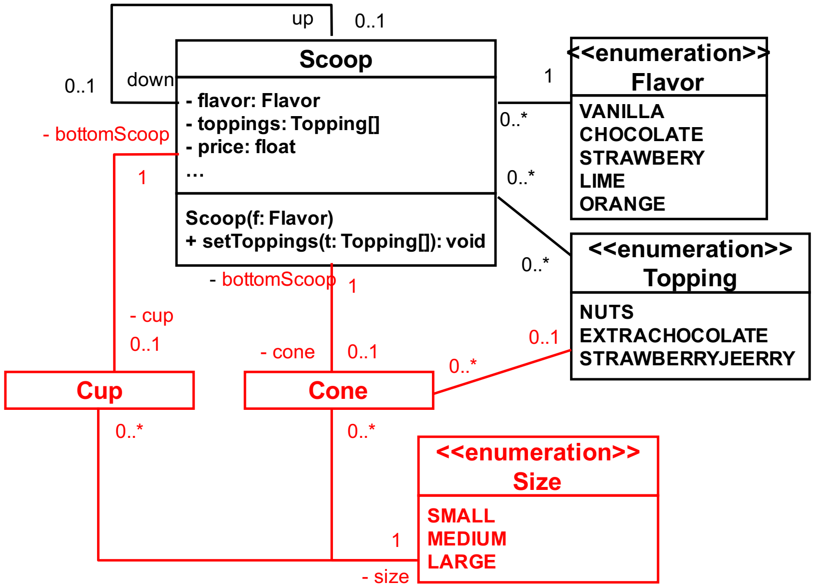
nuts: Topping Topping.NUTS

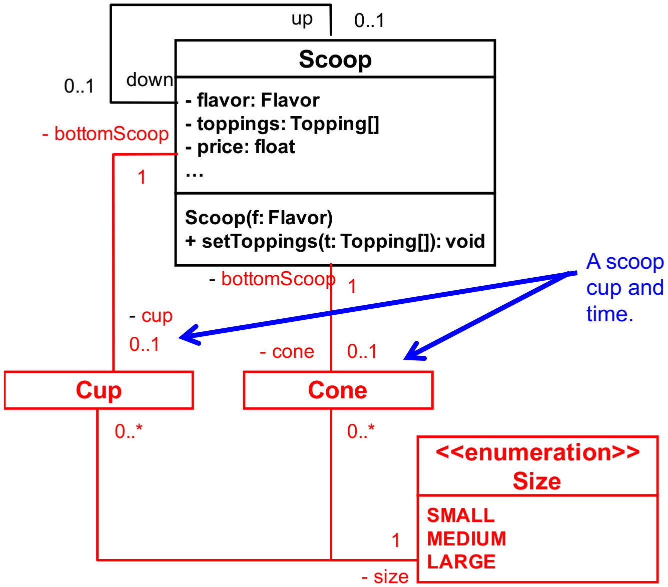
chocolate: Flavor Flavor.CHOCOLATE

strawberry: Flavor Flavor.STRAWBERY

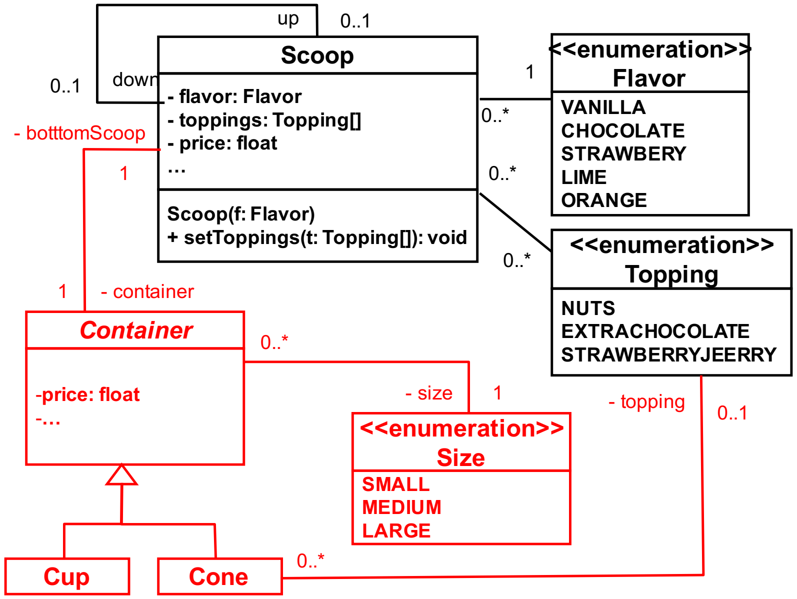
How about these cases?

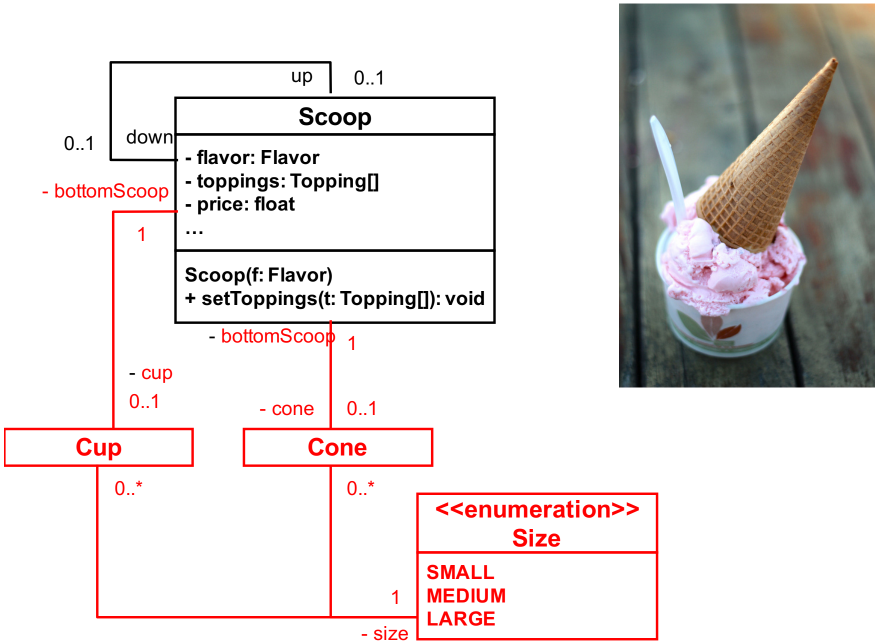
Cone or cup
Size of cone/cup



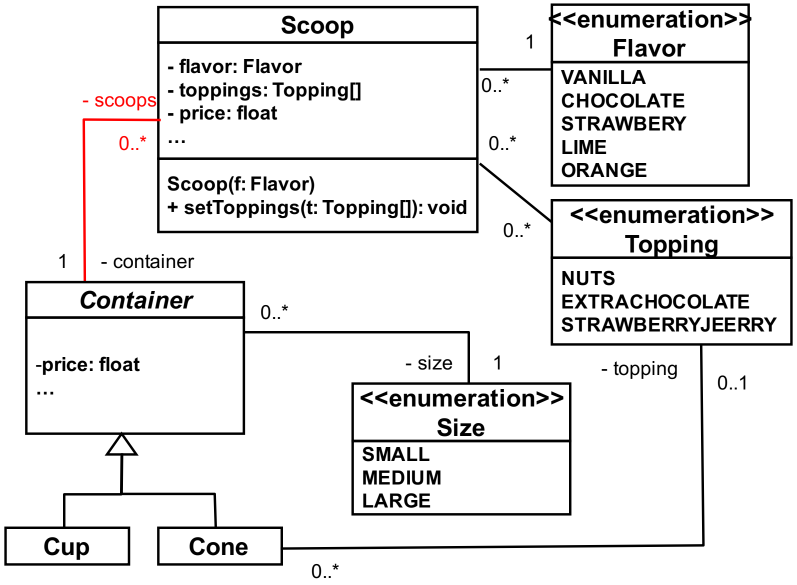


A scoop could sit on both a cup and a cone at the same time.





Now, how about this?



Scoop
- flavor: Flavor
- toppings: Topping[]
- price: float
…

Scoop(f: Flavor)
+ setToppings(t: Topping[]): void

<<enumeration>>
Flavor
VANILLA
CHOCOLATE
STRAWBERY
LIME
ORANGE

<<enumeration>>
Topping
NUTS
EXTRACHOCOLATE
STRAWBERRYJEERRY

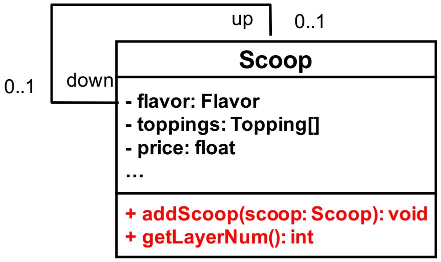
Container
-price: float
…

<<enumeration>>
Size
SMALL
MEDIUM
LARGE

Cup

Cone

Max # of Scoops



Scoop
- flavor: Flavor
- toppings: Topping[]
- price: float
…

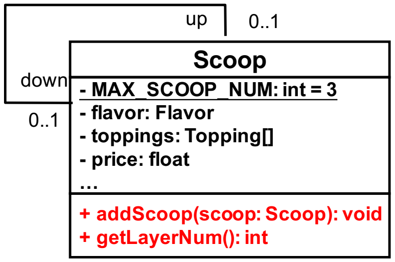
+ addScoop(scoop: Scoop): void
+ getLayerNum(): int

On May 16, 2005, Eisenhower Junior High School set a new world record for producing the ***World's Tallest Ice Cream Cone.*** Three 9th Grade students produced an ice cream cone that reached a height of 13.00 inches.

```
if( getLayerNum() > 3 ){
   // do some error handling }
```

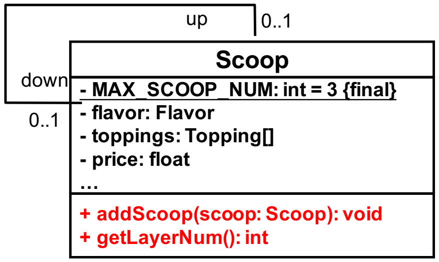
Do not use *magic numbers* directly in your code! Use *symbolic constants* instead to improve the readability of your code.

```
private static final int MAX_SCOOP_NUM = 3;
if( getLayerNum() > MAX_SCOOP_NUM ){
   // do some error handling }
```



Scoop
- MAX_SCOOP_NUM: int = 3
- flavor: Flavor
- toppings: Topping[]
- price: float
…

+ addScoop(scoop: Scoop): void
+ getLayerNum(): int



Scoop
- MAX_SCOOP_NUM: int = 3 {final}
- flavor: Flavor
- toppings: Topping[]
- price: float
…

+ addScoop(scoop: Scoop): void
+ getLayerNum(): int

```
private static final int MAX_SCOOP_NUM = 3;
```

Private or public?

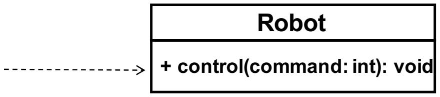
Static or non-static

Final or non-final?

Another Example: Replacing a Magic Number with a Symbolic Constant

Robot controller code

```
Robot r = new Robot();
r.control(0);
r.control(1);
r.control(2);
```

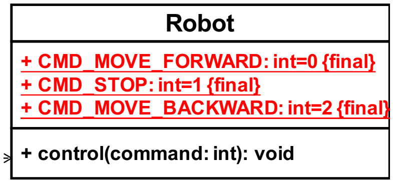

Robot
+ control(command: int): void

```
if( command == 0 )
  // move forward
if( command == 1 )
  // stop
if( command == 2 )
  // move backward
```

- Magic numbers as commands to control a robot.

Robot controller code

```
Robot r = new Robot();
r.control(Robot.CMD_MOVE_FORWARD);
r.control(Robot.CMD_STOP);
r.control(Robot.CMD_MOVE_BACKWARD);
```


Robot
+ CMD_MOVE_FORWARD: int=0 {final}
+ CMD_STOP: int=1 {final}
+ CMD_MOVE_BACKWARD: int=2 {final}
+ control(command: int): void

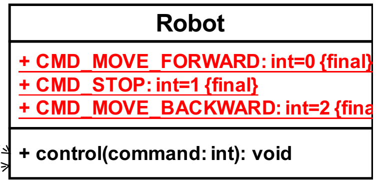
```
if( command == CMD_MOVE_FORWARD )
  // move forward
if( command == CMD_STOP )
  // stop
if( command == CMD_MOVE_BACKWARD )
  // move backward
```

- Replace magic numbers with *public static final* constants.

A Potential Issue

Robot controller code

```
Robot r = new Robot();
r.control(Robot.CMD_MOVE_FORWARD);
r.control(Robot.CMD_STOP);
r.control(Robot.CMD_MOVE_BACKWARD);
```


Robot
+ CMD_MOVE_FORWARD: int=0 {final}
+ CMD_STOP: int=1 {final}
+ CMD_MOVE_BACKWARD: int=2 {fina
+ control(command: int): void

Robot controller code

```
Robot r = new Robot();
r.control(0);
r.control(1);
r.control(2);
```

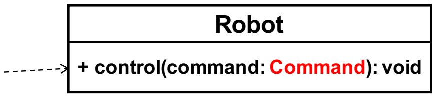
```
if( command == CMD_MOVE_FORWARD )
  // move forward
if( command == CMD_STOP )
  // stop
if( command == CMD_MOVE_BACKWARD )
  // move backward
```

- Clients of Robot can pass integer values to control() by skipping static final constants.

A Solution: Use an Enumeration

Robot controller code

```
Robot r = new Robot();
r.control(Command.MOVE_FORWARD);
r.control(Command.STOP);
r.control(Command.MOVE_BACKWARD);
```


Robot
+ control(command: Command): void

```
if( command == Command.MOVE_FORWARD )
  // move forward
if( command == Command.STOP )
  // stop
if( command == Command.MOVE_BACKWARD )
  // move backward
```

HW 3-1

- Learn Java’s enumeration
  - if you don’t know what it is and how to use it.
- Write Java code for the class diagram in Slide #29. Write test code that makes some ice creams.
  - Use Java’s enumeration
  - Define addScoop() and getLayerNum(), and use a symbolic constant.
    - c.f. Slide #33

HW

- Replace magic number with symbolic constant
  - http://www.refactoring.com/catalog/replaceMagicNumberWithSymbolicConstant.html
  - http://sourcemaking.com/refactoring/replace-magic-number-with-symbolic-constant

CS682

- Will have 2 sections
  - One taught/advised by me
  - the other taught/advised by another instructor