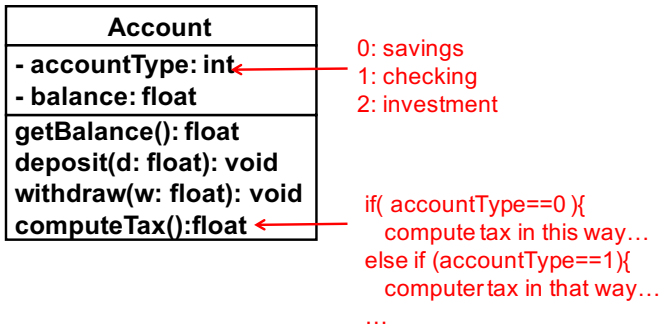
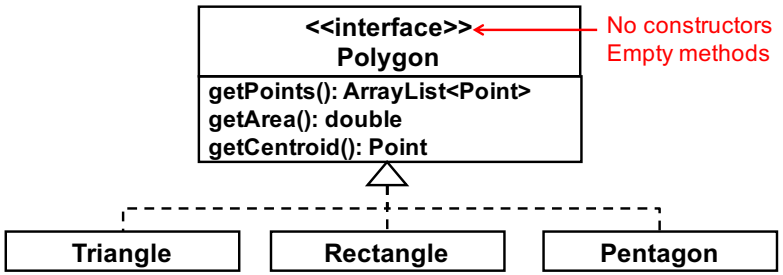
Your Email Address

- Send your (preferred) email address to umasscs680@gmail.com ASAP.
  - I will use that address to email you lecture notes, announcements, etc.

If Polymorphism is not available…



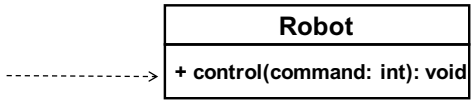



```
ArrayList<Polygon> p = new ArrayList<Polygon>();
p.add( new Triangle( new Point(0,0),
                     new Point(2,2),
                     new Point(1,3) ));
p.add( new Rectangle ( new Point(0,0)... ));
Iterator<Polygon> it = p.iterator();
while( it.hasNext() ){
  Polygon nextP = it.next();
  System.out.println( nextP.getPoints() );
  System.out.println( nextP.getArea() );
  System.out.println( nextP.getCentroid() ); }
```


Replacing a Magic Number with a Symbolic Constant

Robot controller code

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

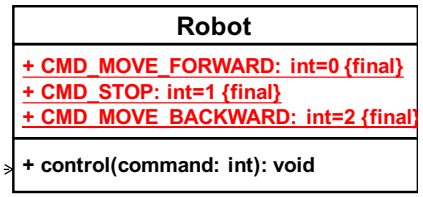



```
if( command == 0 )
  // move forward
if( command == 1 )
  // stop
if( command == 2 )
  // move backward
```

- 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);
```



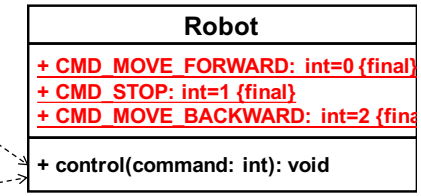

```
if( command == CMD_MOVE_FORWARD )
  // move forward
if( command == CMD_STOP )
  // stop
if( command == CMD_MOVE_BACKWARD )
  // move backward
```

- Do not use ***magic numbers*** directly in your code! Use ***symbolic constants*** instead to improve the readability and maintainability of your code
  - *static final* constants.
    - `public static final int CMD_MOVE_FORWARD = 0;`

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 controller code

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

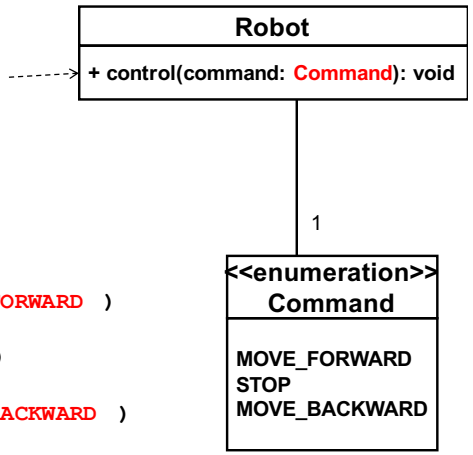
```
if( command == CMD_MOVE_FORWARD )
  // move forward
if( command == CMD_STOP )
  // stop
if( command == CMD_MOVE_BACKWARD )
  // move backward
```

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

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);
```




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


Note

- Learn Java's enumeration
  - if you don't know what it is and how to use it.

HW

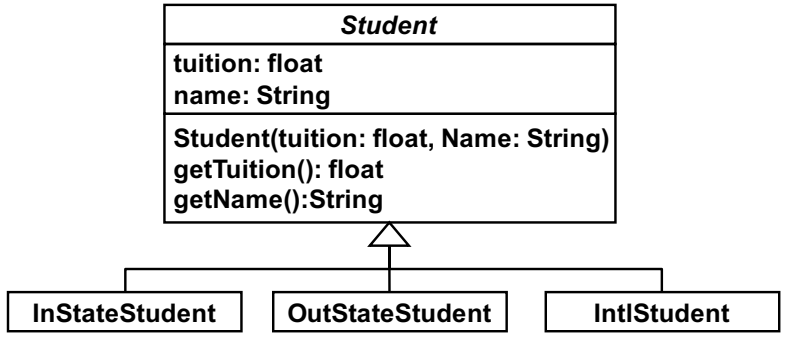
- Learn general ideas on refactoring
  - Refactoring = Restructuring existing code by revising its internal structure without changing its external behavior.
    - http://en.wikipedia.org/wiki/Refactoring
    - http://www.refactoring.com/
    - http://sourcemaking.com/refactoring
    - *Refactoring: Improving the Design of Existing Code*
      - by Martin Fowler
      - Addison-Wesley
- Read "Replace Conditional with Polymorphism"
  - http://www.refactoring.com/catalog/replaceConditionalWithPolymorphism.html
  - http://sourcemaking.com/refactoring/replace-conditional-with-polymorphism

HW

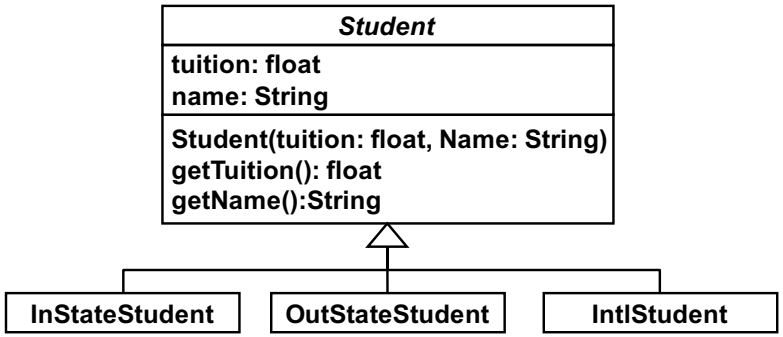
- Read "Replace magic number with symbolic constant"
  - http://www.refactoring.com/catalog/replaceMagicNumberWithSymbolicConstant.html
  - http://sourcemaking.com/refactoring/replace-magic-number-with-symbolic-constant

When to Use Inheritance and When not to Use it

An Inheritance Example



- In-state, out-state and int'l students are students.
  - "Is-a" relationship
  - Conceptually, there are no problems.
- A class inheritance is NOT reasonable if subclass instances may want to dynamically change their classes (i.e. student status) in the future.



- An out-state student can be eligible to become an in-state student after living in MA for some years.
- An int'l student can become an in/out-state student through some visa status change.

Dynamic Class Change

- Most programming languages do not allow this.
  - Exceptions: CLOS and a few scripting languages

– Need to create a new class instance and copy "some" existing data field values to it.

```
IntlStudent intlStudent = new IntlStudent(…);
new OutStateStudent( intlStudent.getTuition(),
                     intlStudent.getName() );
```

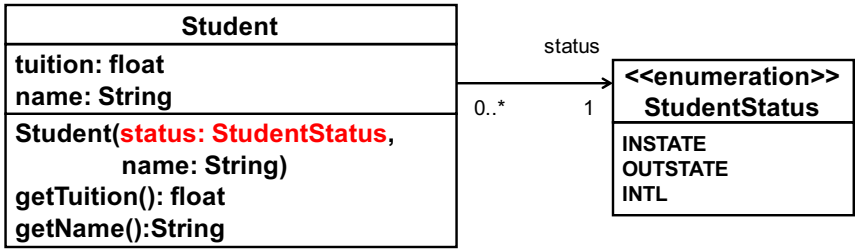
  - Not all existing data field values may go to a new instance.
    – e.g. Data specific to int'l students such as I-20 number and visa #

– Need a "deep" copy if an instance in question is connected with other instances.
  - e.g., IntlStudent → Address

When to Use an Inheritance?

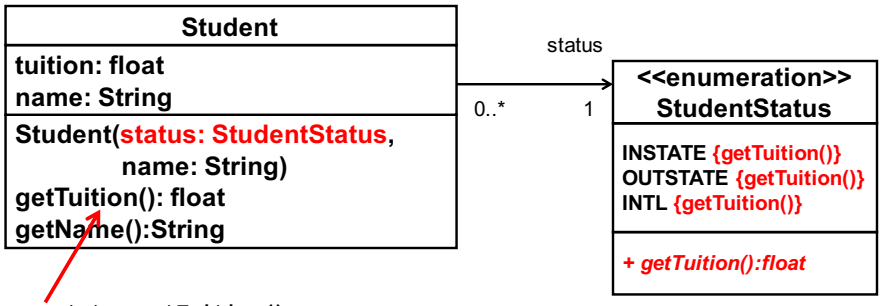
- An "is-a" relationship exists between two classes.
- No instances change their classes dynamically.
- No instances belong to more than two classes.

Enumeration?



- Can allow student status changes
- Need to have conditional statements in getTuition()
  – There are two ways to remove the conditional statements.
    - With extra methods in an enum
    - With extra classes (*State* design pattern)

Alternative Design #1

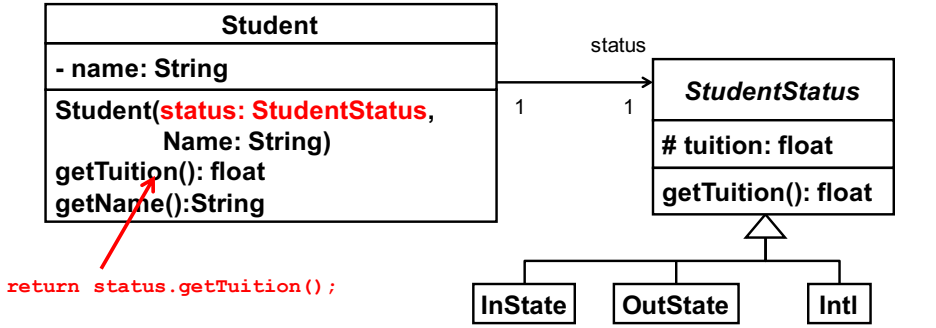



```
return status.getTuition();
```

```
Student s1 =
  new Student(StudentStatus.INSTATE,
              "John Smith" );
s1.getTuition();
```

```
public emum StudentCategory{
 INSTATE{
  @Override public void getTuition(){
    return 1000;
  }
 }
 OUTSTATE{
  @Override public void getTuition(){
    return 2000;
  }
 }
 ...
}
```


Alternative Design #2



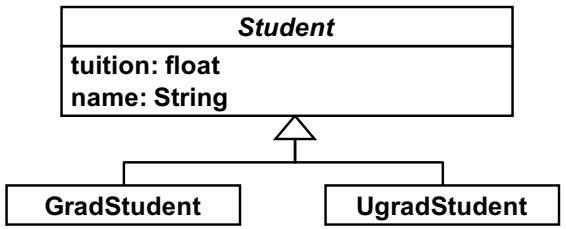

```
return status.getTuition();
```

```
Student s1 = new Student( new InState(1000), "John Smith" );
s1.getTuition();
```


HW 3

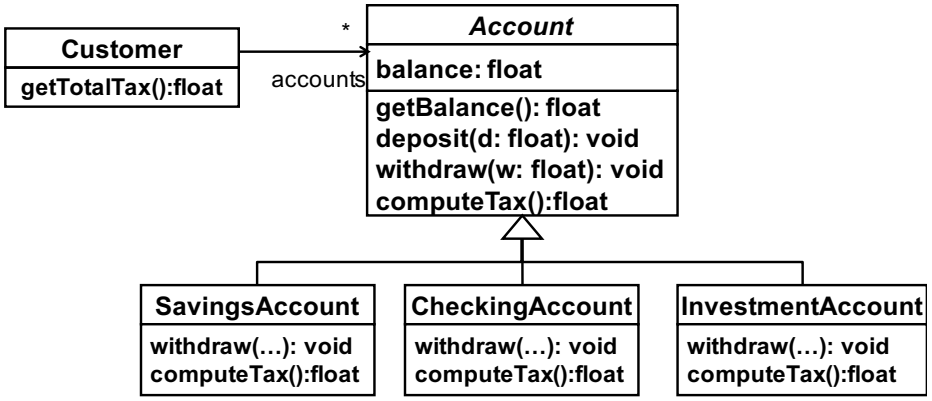
- Implement either of the two alternative designs (#1 or #2)
- [OPTIONAL] If interested, implement both. You will get extra points.

Another Example

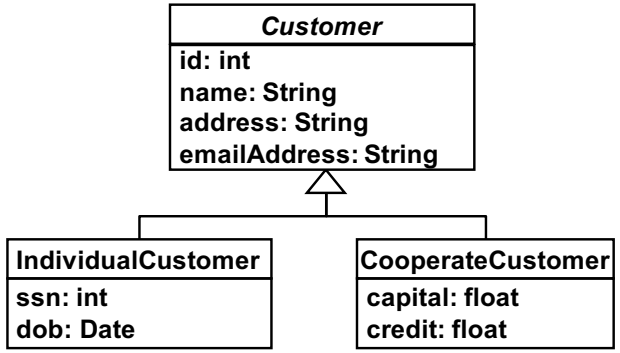


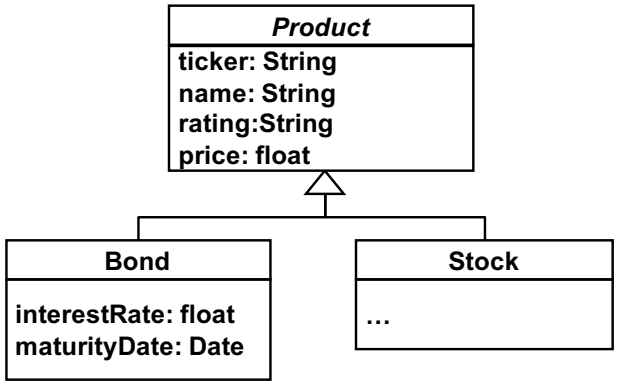
- Grad and u-grad students are students.
  - "Is-a" relationship
  - Conceptually, no problem.
- A class inheritance is NOT reasonable if subclass instances may want to dynamically change their classes in the future.
  - Implementation limitation: Most programming languages do not allow this.

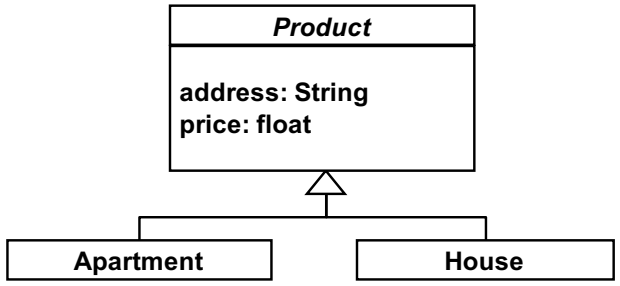
More Examples

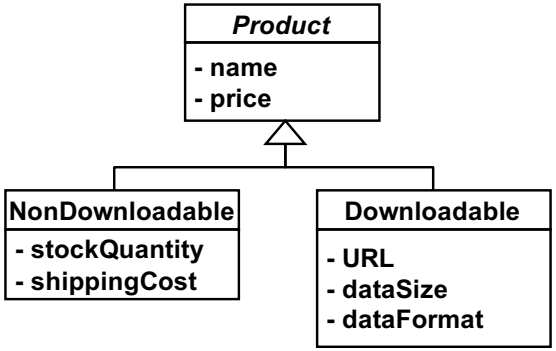


- An Account instance needs to change its type?
  - Savings to checking? No.



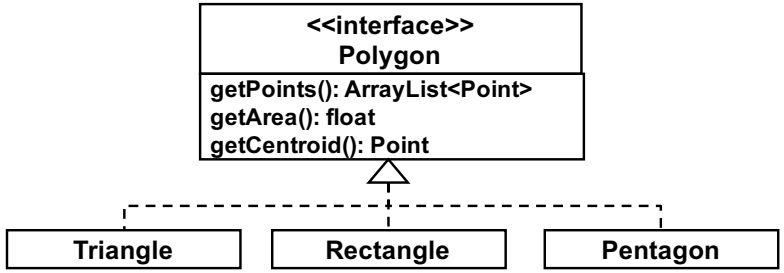




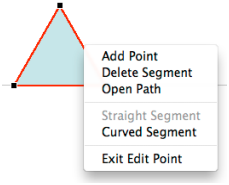


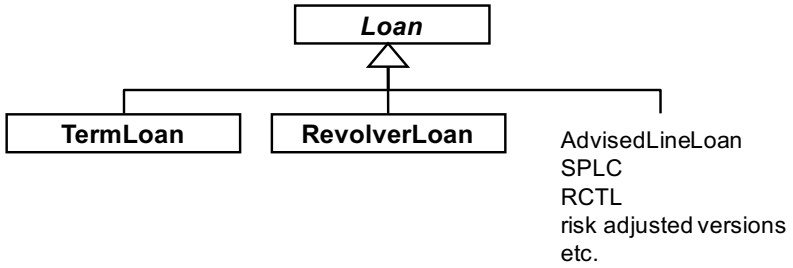
- Assume a product sales app at an online retail store (e.g., Amazon)
- Does this inheritance-based design make sense?
  - An is-a relationship between the super class and a subclass?
  - Does a subclass instance need to change its class?

Some More Examples

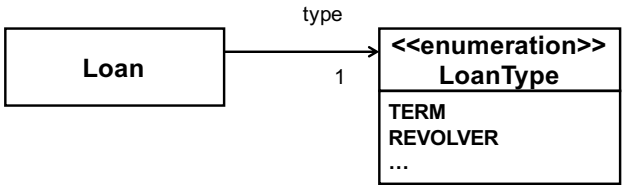


- Can a triangle become a rectangle?
- Do we allow that?
  - Maybe, depending on requirements.

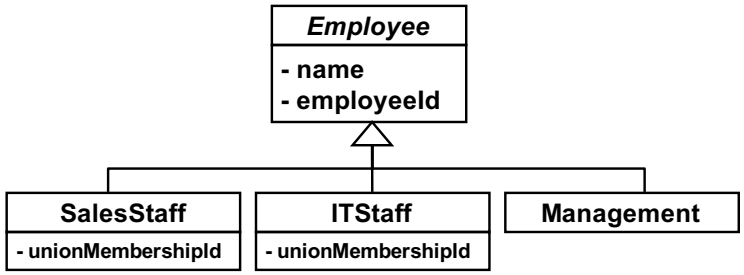




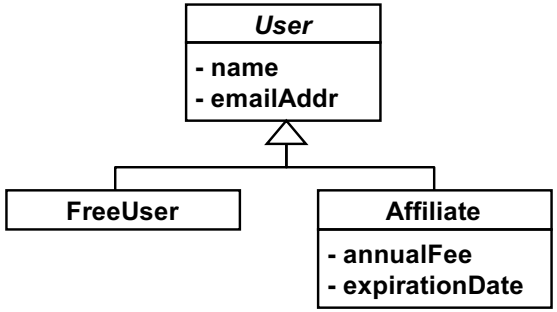
- Term loan
  - Must be fully paid by its maturity date.
- Revolver
  - e.g. credit card
  - With a spending limit and expiration date
- A revolver can transform into a term loan when it expires.



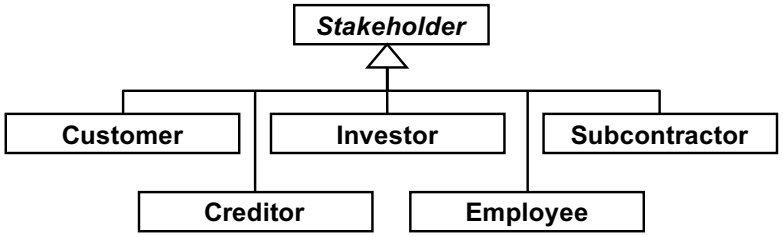
- Use an enumeration
- Use the *State* design pattern



- How about this?
- Assume an employee management system.



- How about this?
- Assume a user management system
  - c.f. Amazon (regular users v.s. Amazon Prime users), Dropbox, Google Drive, etc.



- An employee can be a customer and/or an investor.
- A subcontractor can be a customer.
- If an instance belongs to two or more classes, do not use inheritance relationships.

When to Use an Inheritance?

- An “is-a” relationship exists between two classes.
- No instances change their classes dynamically.
- No instances belong to more than two classes.

HW: Read these pages.

- Replace Type Code with Class
  - http://www.refactoring.com/catalog/replaceTypeCodeWithClass.html
  - http://sourcemaking.com/refactoring/replace-type-code-with-class
- Replace Type Code with State/Strategy
  - http://www.refactoring.com/catalog/replaceTypeCodeWithStateStrategy.html
  - http://sourcemaking.com/refactoring/replace-type-code-with-state-strategy
- Replace Type Code with Subclasses
  - http://www.refactoring.com/catalog/replaceTypeCodeWithSubclasses.html
  - http://sourcemaking.com/refactoring/replace-type-code-with-subclasses