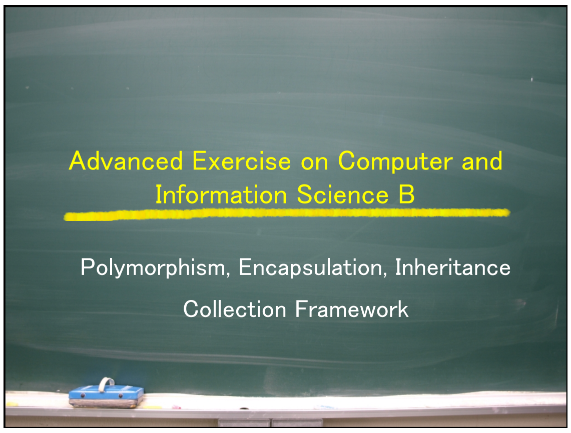
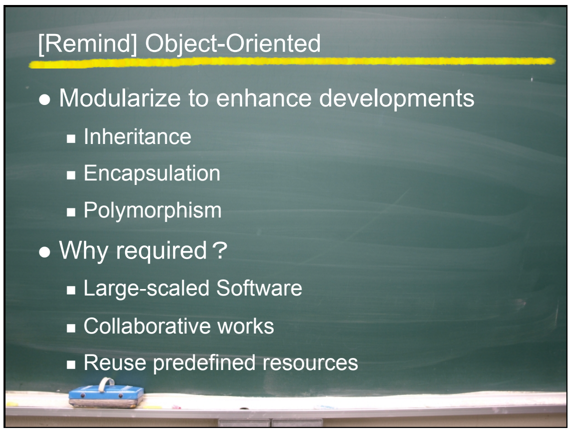


Advanced Exercise on Computer and Information Science B

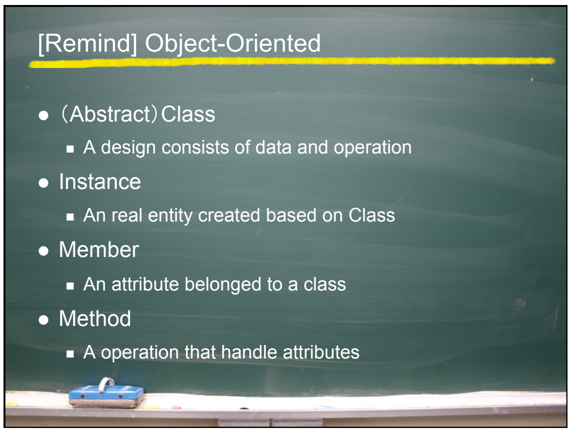
Polymorphism, Encapsulation, Inheritance

Collection Framework



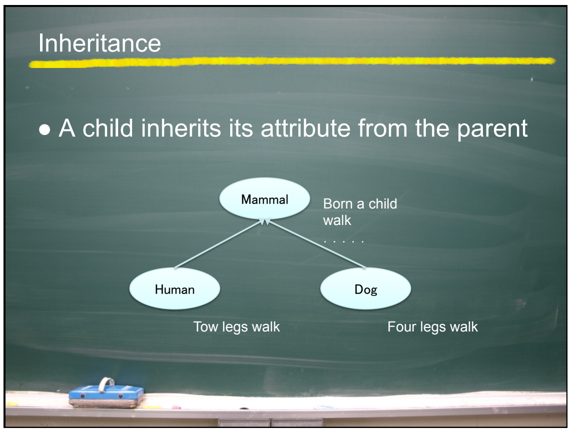
[Remind] Object-Oriented

- Modularize to enhance developments
  - Inheritance
  - Encapsulation
  - Polymorphism
- Why required？
  - Large-scaled Software
  - Collaborative works
  - Reuse predefined resources



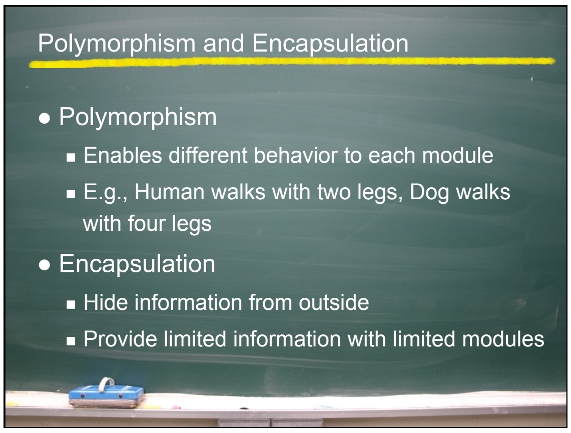
[Remind] Object-Oriented

- （Abstract）Class
  - A design consists of data and operation
- Instance
  - An real entity created based on Class
- Member
  - An attribute belonged to a class
- Method
  - A operation that handle attributes



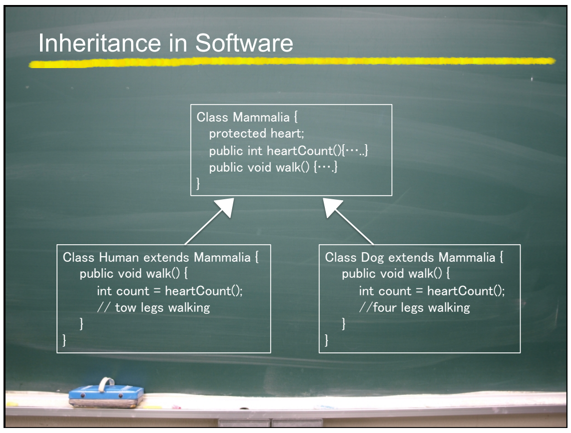
Inheritance

- A child inherits its attribute from the parent



Polymorphism and Encapsulation

- Polymorphism
  - Enables different behavior to each module
  - E.g., Human walks with two legs, Dog walks with four legs
- Encapsulation
  - Hide information from outside
  - Provide limited information with limited modules

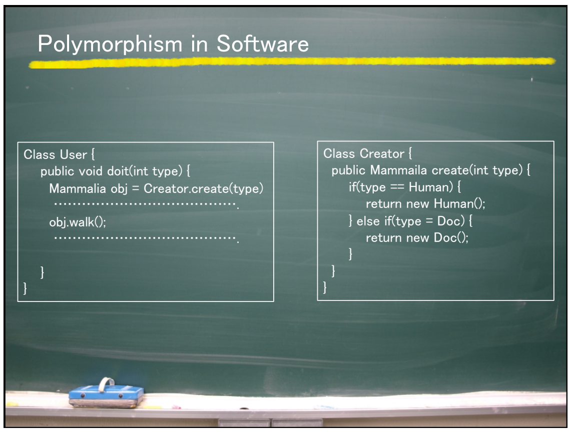


Inheritance in Software

```
Class Mammalia {
   protected heart;
   public int heartCount(){･･･..}
   public void walk() {･･･.}
}
```

```
Class Human extends Mammalia {
    public void walk() {
        int count = heartCount();
        // tow legs walking
    }
}
```

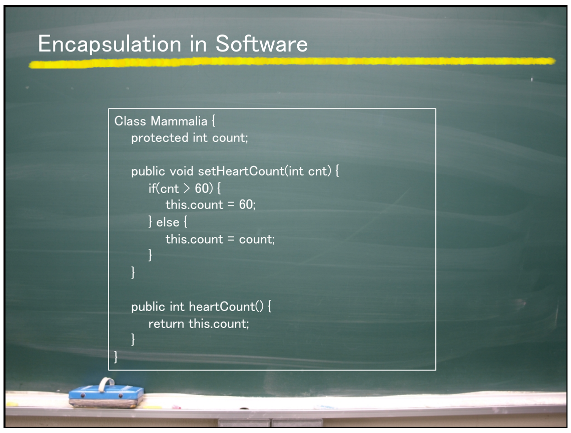
```
Class Dog extends Mammalia {
    public void walk() {
        int count = heartCount();
        //four legs walking
    }
}
```



Polymorphism in Software

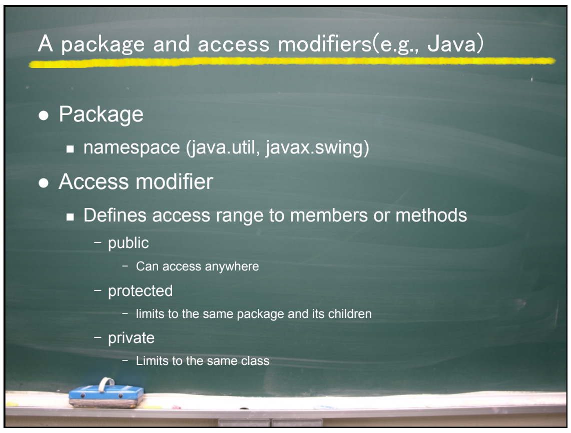
```
Class User {
    public void doit(int type) {
      Mammalia obj = Creator.create(type)
       ………………………………….
      obj.walk();
       ………………………………….

    }
}
```

```
Class Creator {
  public Mammaila create(int type) {
      if(type == Human) {
          return new Human();
      } else if(type = Doc) {
          return new Doc();
      }
  }
}
```



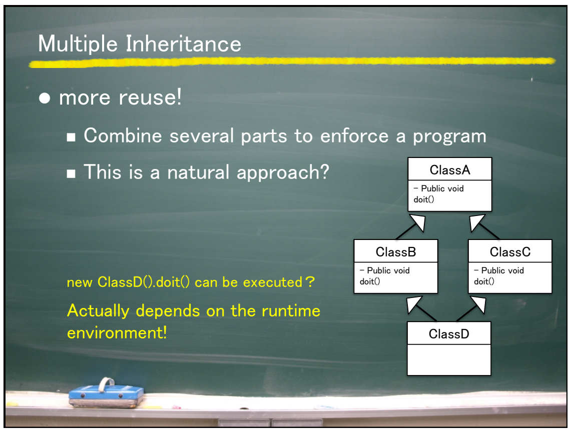
Encapsulation in Software

```
Class Mammalia {
    protected int count;

    public void setHeartCount(int cnt) {
        if(cnt > 60) {
            this.count = 60;
        } else {
            this.count = count;
        }
    }

    public int heartCount() {
        return this.count;
    }
}
```



A package and access modifiers(e.g., Java)

- Package
  - namespace (java.util, javax.swing)
- Access modifier
  - Defines access range to members or methods
    - public
      - Can access anywhere
    - protected
      - limits to the same package and its children
    - private
      - Limits to the same class

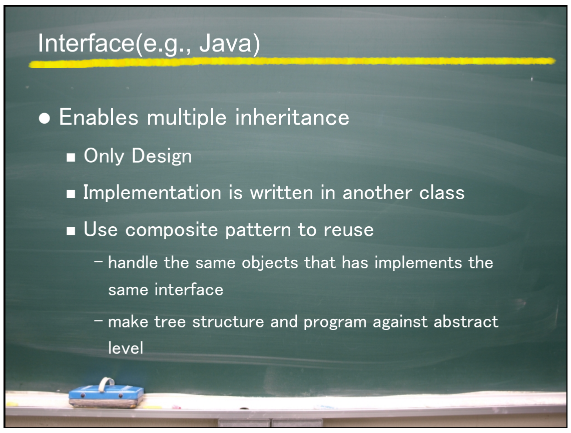


Multiple Inheritance

- more reuse!
  - Combine several parts to enforce a program
  - This is a natural approach?

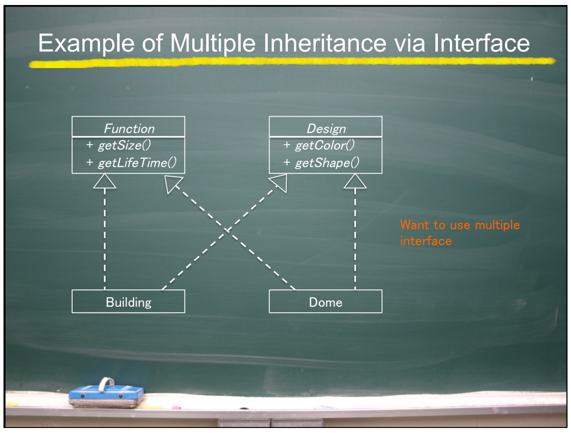
new ClassD().doit() can be executed？

Actually depends on the runtime environment!



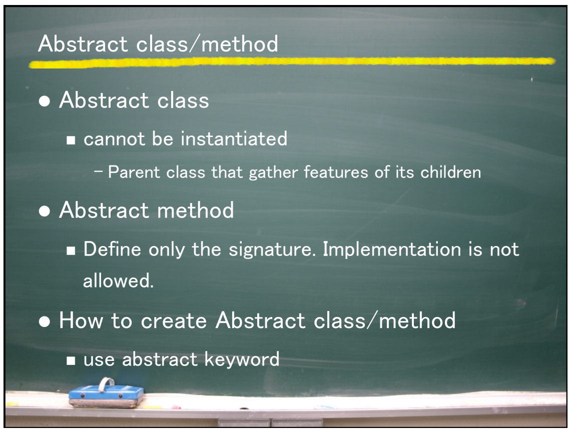
Interface(e.g., Java)

- Enables multiple inheritance
  - Only Design
  - Implementation is written in another class
  - Use composite pattern to reuse
    - handle the same objects that has implements the same interface
    - make tree structure and program against abstract level



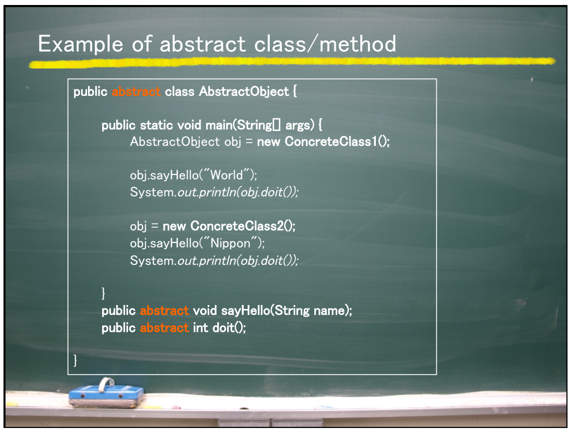
Example of Multiple Inheritance via Interface

Want to use multiple interface



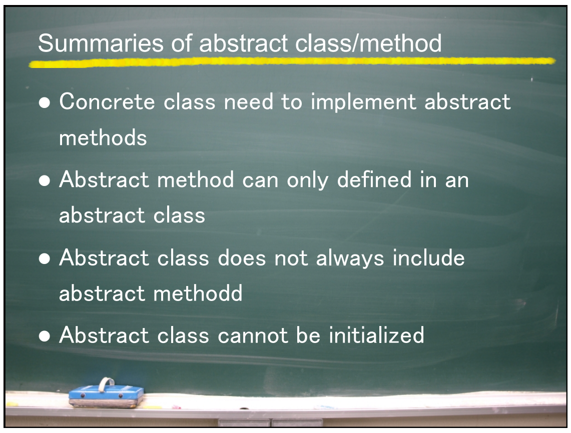
Abstract class/method

- Abstract class
  - cannot be instantiated
    - Parent class that gather features of its children
- Abstract method
  - Define only the signature. Implementation is not allowed.
- How to create Abstract class/method
  - use abstract keyword



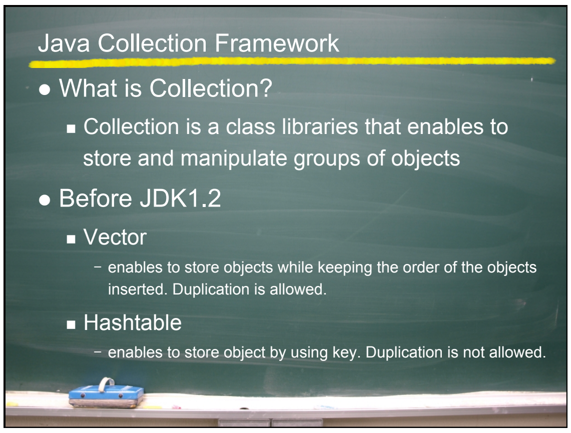
Example of abstract class/method

```
public abstract class AbstractObject {

	public static void main(String[] args) {
		AbstractObject obj = new ConcreteClass1();

		obj.sayHello("World");
		System.out.println(obj.doit());

		obj = new ConcreteClass2();
		obj.sayHello("Nippon");
		System.out.println(obj.doit());

	}
	public abstract void sayHello(String name);
	public abstract int doit();

}
```



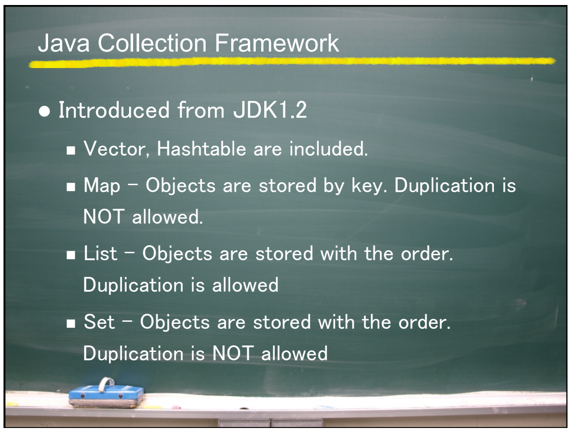
Summaries of abstract class/method

- Concrete class need to implement abstract methods
- Abstract method can only defined in an abstract class
- Abstract class does not always include abstract methodd
- Abstract class cannot be initialized



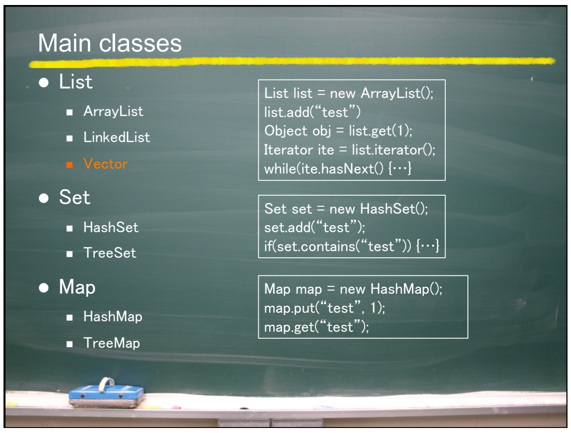
Java Collection Framework

- What is Collection?
  - Collection is a class libraries that enables to store and manipulate groups of objects
- Before JDK1.2
  - Vector
    - enables to store objects while keeping the order of the objects inserted. Duplication is allowed.
  - Hashtable
    - enables to store object by using key. Duplication is not allowed.



Java Collection Framework

- Introduced from JDK1.2
  - Vector, Hashtable are included.
  - Map – Objects are stored by key. Duplication is NOT allowed.
  - List – Objects are stored with the order. Duplication is allowed
  - Set – Objects are stored with the order. Duplication is NOT allowed



Main classes

- List
  - ArrayList
  - LinkedList
  - Vector

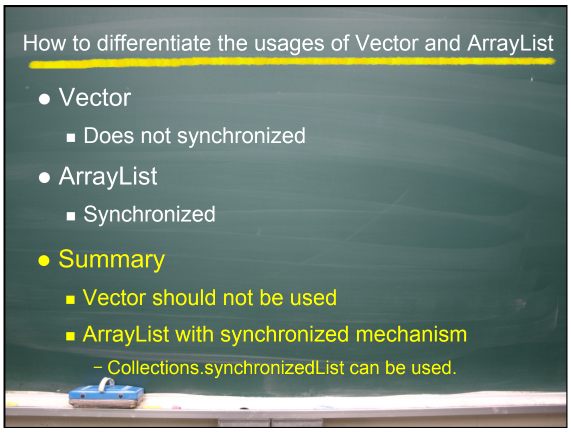
```
List list = new ArrayList();
list.add("test")
Object obj = list.get(1);
Iterator ite = list.iterator();
while(ite.hasNext() {…}
```

- Set
  - HashSet
  - TreeSet

```
Set set = new HashSet();
set.add("test");
if(set.contains("test")) {…}
```

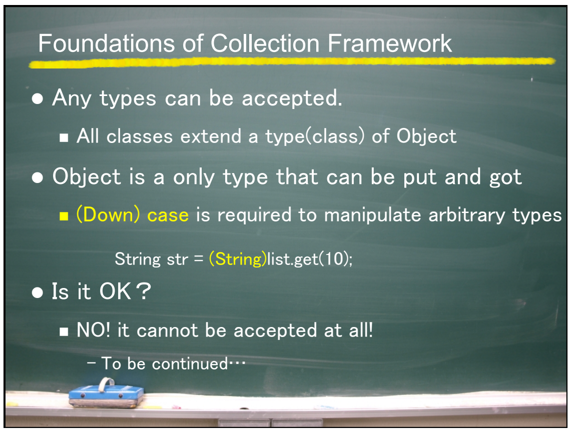
- Map
  - HashMap
  - TreeMap

```
Map map = new HashMap();
map.put("test", 1);
map.get("test");
```



How to differentiate the usages of Vector and ArrayList

- Vector
  - Does not synchronized
- ArrayList
  - Synchronized
- Summary
  - Vector should not be used
  - ArrayList with synchronized mechanism
    - Collections.synchronizedList can be used.



Foundations of Collection Framework

- Any types can be accepted.
  - All classes extend a type(class) of Object
- Object is a only type that can be put and got
  - (Down) case is required to manipulate arbitrary types

```
String str = (String)list.get(10);
```

- Is it OK？
  - NO! it cannot be accepted at all!
    - To be continued…