




Jin L.C. Guo

M2 (c) - Types and Polymorphism

Image Source: https://upload.wikimedia.org/wikipedia/commons/2/2b/Cepaea_nemoralis_active_pair_on_tree_trunk.jpg

Summary so far

- Programming mechanism:
Java Generics, Java Nested Classes
- Concepts and Principles:
Separation of concerns;
- Patterns and Antipatterns:
STRATEGY, SWITCH Statement 
- Design techniques:
Function objects

Objective of the rest of the module

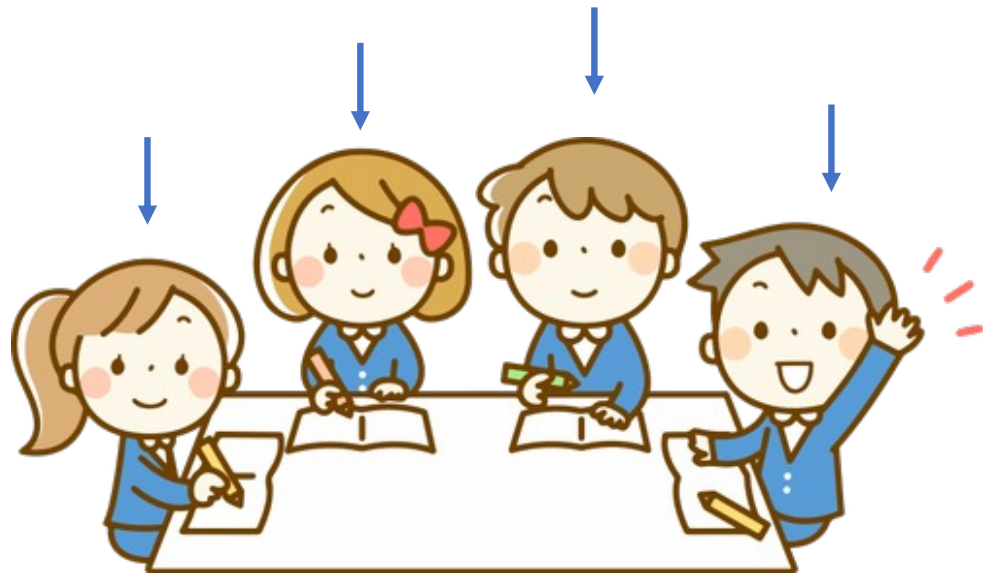
- Concepts and Principles:
Interface Segregation Principle

- Patterns and Antipatterns:

ITERATOR

How to traverse students enrolled in the class?

- So that
 - I can add grade to each student
 - I can print each student's ID
 - I can ...



Activity: How to allow the client code to traverse students enrolled in the class?

```
public class Course
{
    private List<Student> aEnrollment
        = new ArrayList<>();

    ... ..

    public List<Student> getStudents()
    {
        return Collections.unmodifiableList(aEnrollment);
    }

}
```

```
for(int i=0; i<course.getStudents().size; i++)
{
    Student s = course.getStudents().get(i);
    /* do something using Student instance*/
}
```

Can we make the way of traversing the students irrelevant to how the students are stored internally?

What is needed during traversing?

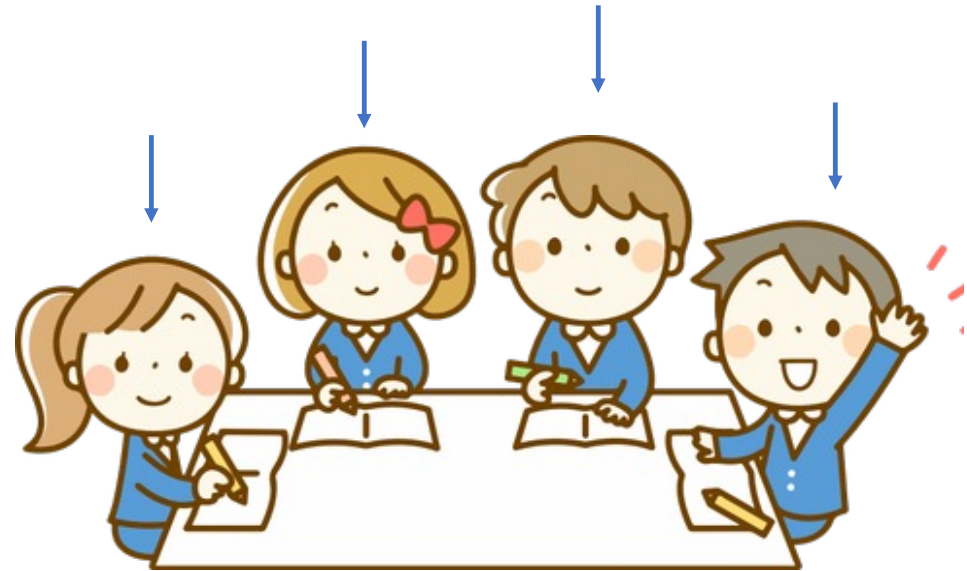
Keep track with the current element and know how to get to the next.

Student next()

```
for(int i=0; i<course.getStudents().size; i++)  
{  
    Student s = course.getStudents().get(i);  
    /* do something using Student instance*/  
}
```

Know if the end has been reached

boolean hasNext()



How to traverse students enrolled in the class?

StudentIterator
Student next() boolean hasNext()

```
for(int i=0; i<course.getStudents().size; i++)  
{  
    Student s = course.getStudents().get(i);  
    /* do something using Student instance*/  
}
```

```
public class Course  
{  
    private List<Student> aEnrollment  
        = new ArrayList<>();  
  
    ... ..  
  
    public List<Student> getStudents()  
    {  
        return Collections.unmodifiableList(aEnrollment);  
    }  
  
}
```

How to traverse students enrolled in the class?

StudentIterator
Student next() boolean hasNext()

```
public class Course
{
    private List<Student> aEnrollment
        = new ArrayList<>();

    ... ..

    public StudentIterator getIterator()
    {
        /* create student iterator*/

        return sIterator;
    }
}
```

```
StudentIterator sIterator = course.getIterator();
while(sIterator.hasNext())
{
    Student s = sIterator.next();
    /* do something using Student instance*/
}
```


Java Iterator Interface

- Interface Iterator<E>

E - the type of elements returned by this iterator

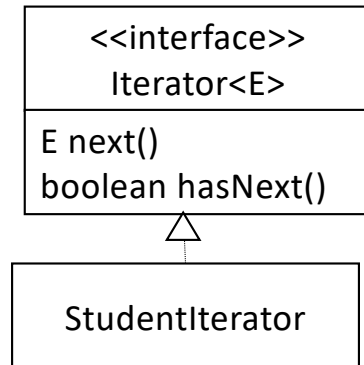
boolean hasNext();

Returns true if the iteration has more elements.

E next();

Returns the next element in the iteration.

How to traverse students enrolled in the class?



```
public class Course
{
    private List<Student> aEnrollment
        = new ArrayList<>();

    ... ..

    public StudentIterator getIterator()
    {
        /* create student iterator*/
        return sIterator;
    }
}
```

```
StudentIterator sIterator = course.getIterator();
while(sIterator.hasNext())
{
    Student s = sIterator.next();
    /* do something using Student instance*/
}
```

Iterator<Student>

Adding even more flexibility: how to traverse students in data type such as Club, Committee, ...?

```
public class Course
{
    private List<Student> aEnrollment
        = new ArrayList<>();

    ... ..

    public Iterator<Student> getIterator()
    {
        /* create student iterator*/
        return sIterator;
    }

}
```

```
Iterator<Student> sIterator = course.getIterator();
while(sIterator.hasNext())
{
    Student s = sIterator.next();
    /* do something using Student instance*/
}
```

Encapsulate Iterable Behavior

- **Java Iterable<T> Interface**

T - the type of elements returned by the iterator

```
public Iterator<T> iterator()
```

Adding even more flexibility

Same client code to traverse students in data type such as Club, Committee, ...

```
public class Course implements Iterable<Student>
{
    private List<Student> aEnrollment
        = new ArrayList<>();

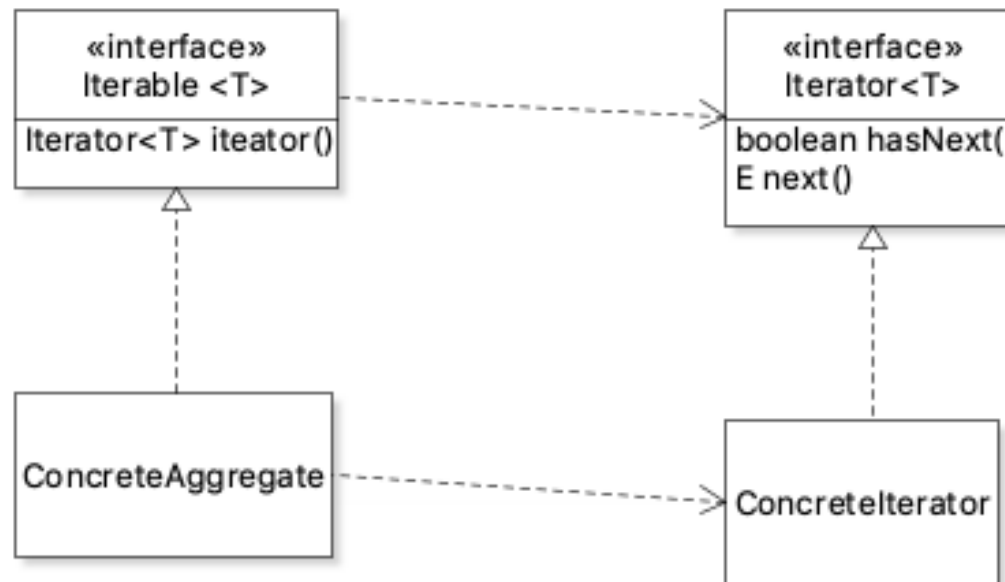
    ... ..

    @Override
    public Iterator<Student> iterator()
    {
        /* create student iterator*/
        return aEnrollment.iterator();
    }
}
```

```
Iterator<Student> sIterator = course.getIterator();
while(sIterator.hasNext())
{
    Student s = sIterator.next();
    /* do something using Student instance*/
}
```

Iterator Design Pattern

- Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation




Adding even more flexibility

Same client code to traverse students in data type such as Club, Committee, ...

```
public class Course implements Iterable<Student>
{
    private List<Student> aEnrollment
        = new ArrayList<>();

    ... ..

    @Override
    public Iterator<Student> iterator()
    {
        /* create student iterator*/
        return aEnrollment.iterator();
    }
}
```



```
Iterator<Student> sIterator = course.getIterator();
while(sIterator.hasNext())
{
    Student s = sIterator.next();
    /* do something using Student instance*/
}
```

Objective of this class

- Concepts and Principles:

Interface Segregation Principle

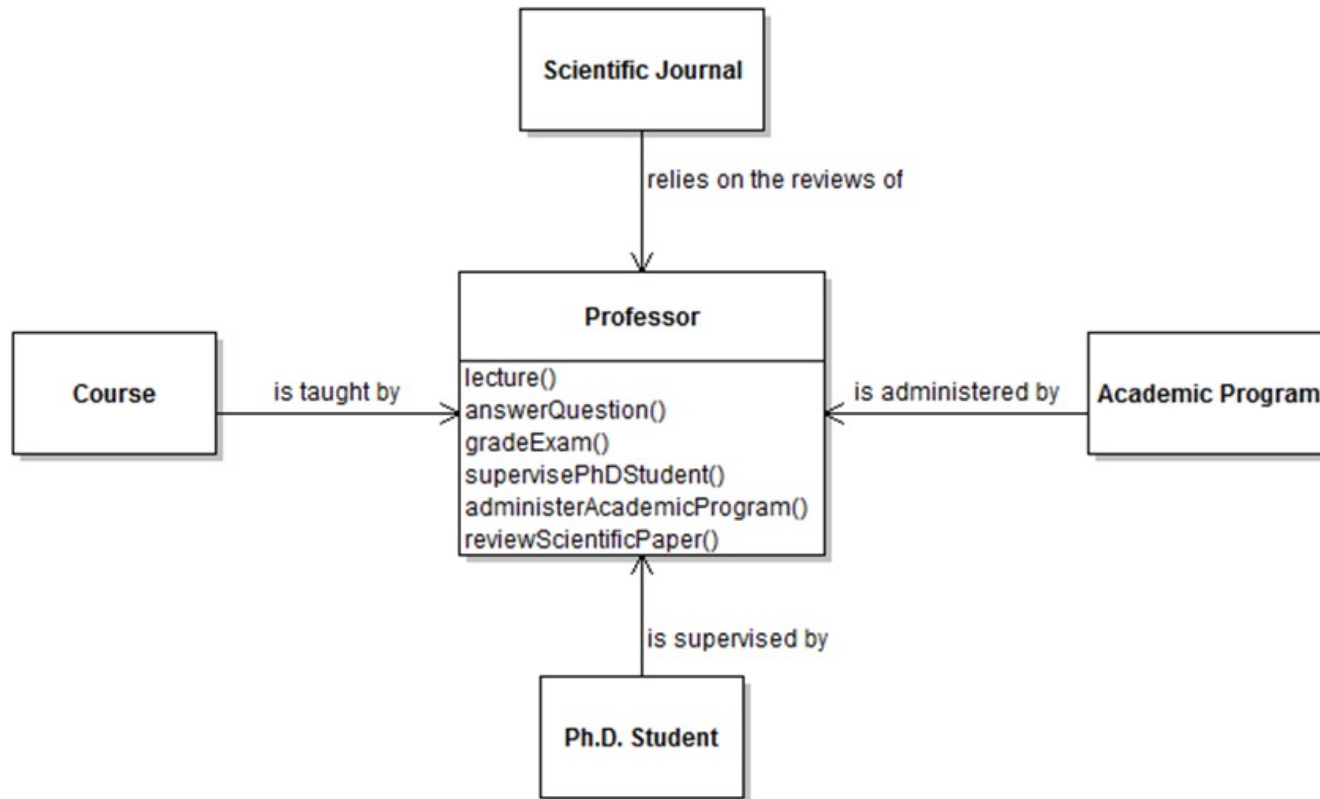
- Patterns and Antipatterns:

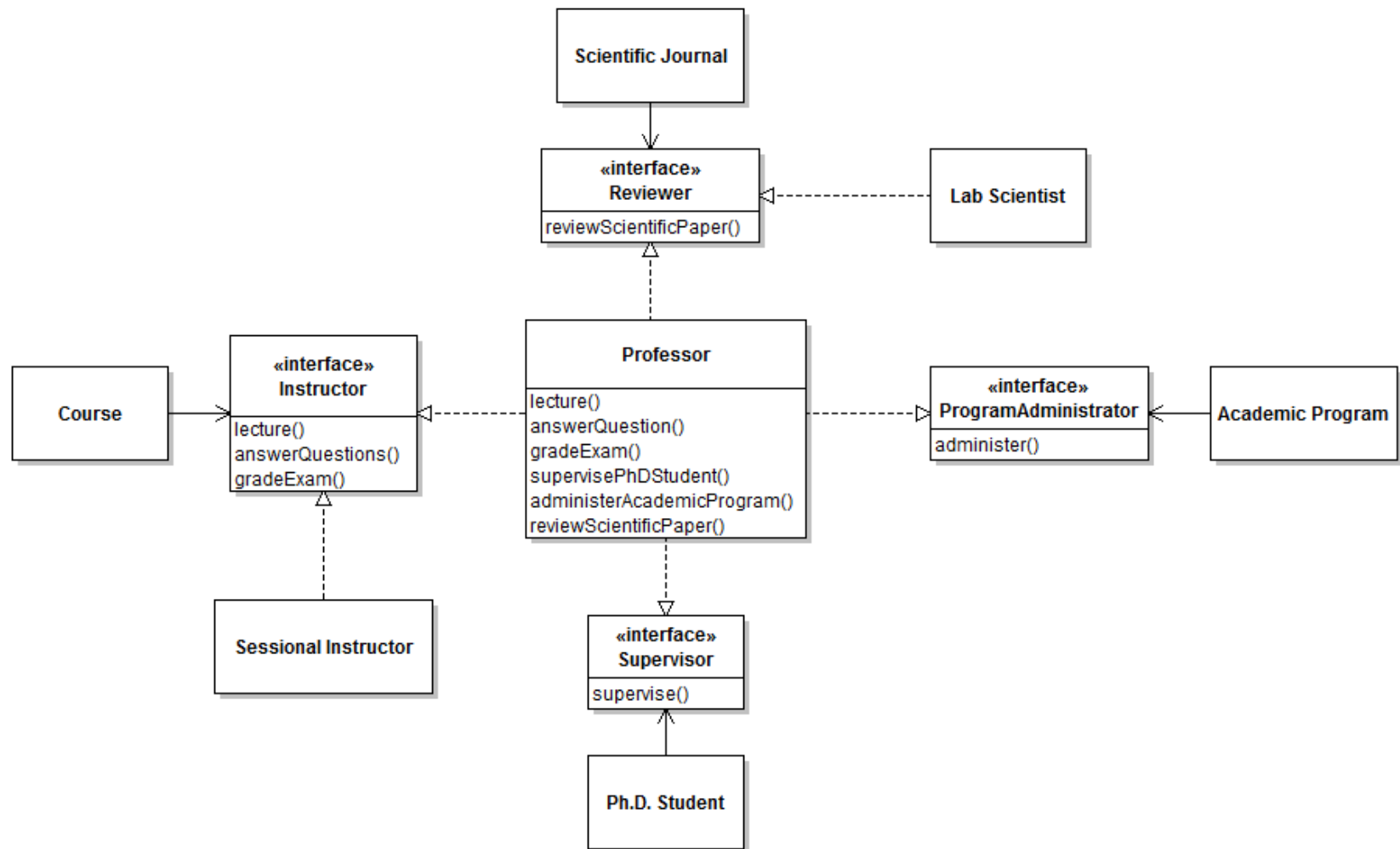
ITERATOR

Interface Segregation Principle

Clients should not be forced to depend on interfaces they do not need.

Interface Segregation Principle





Next: M3 - Object State