Jin L.C. Guo

M2 (a) - Types and Polymorphism

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

Recap of last module

- Programming mechanisms:
  - Scope and Visibility
- Concepts and Principles:
  - Information Hiding, Encapsulation, Escaping Reference, Immutability
- Design Techniques:
  - Object Diagrams
- Patterns and Antipatterns:
  - Primitive Obsession 👎

Objective of this lecture

- Concepts and Principles:

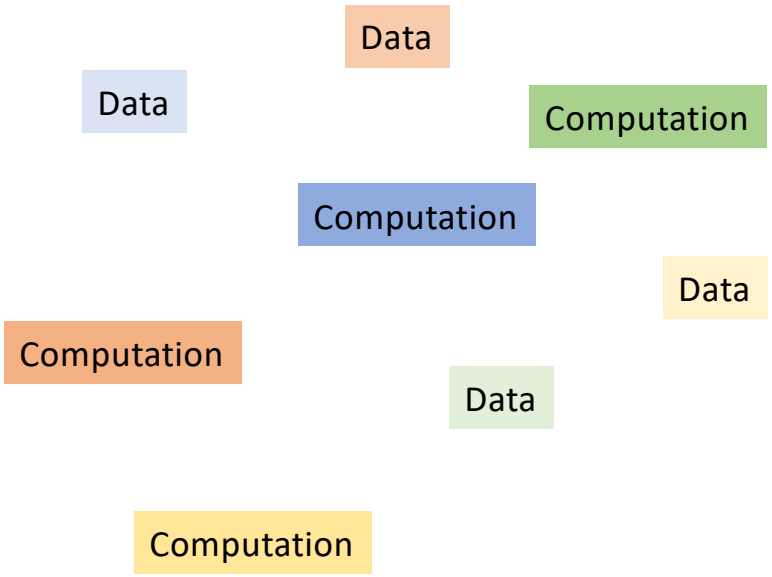
Class's interface, Separation of concerns

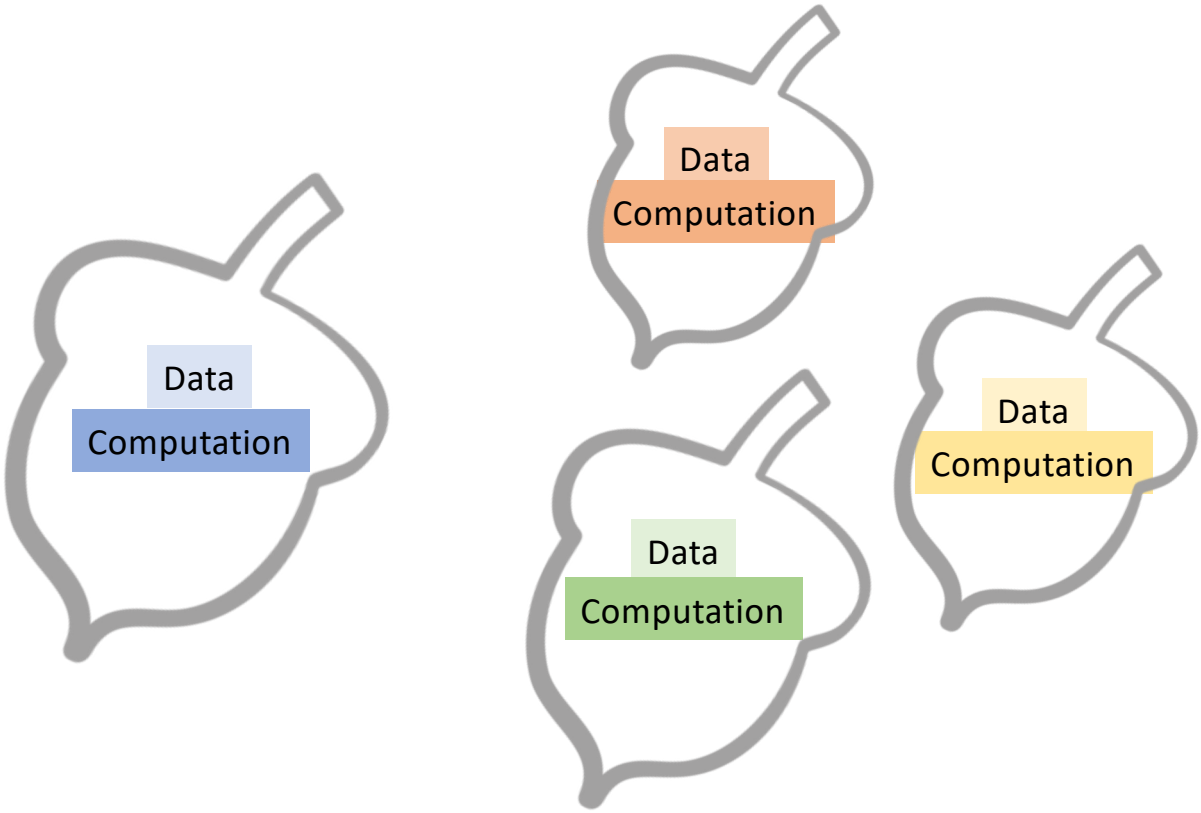
- Programming mechanism:

Java Interface type, Subtype polymorphism

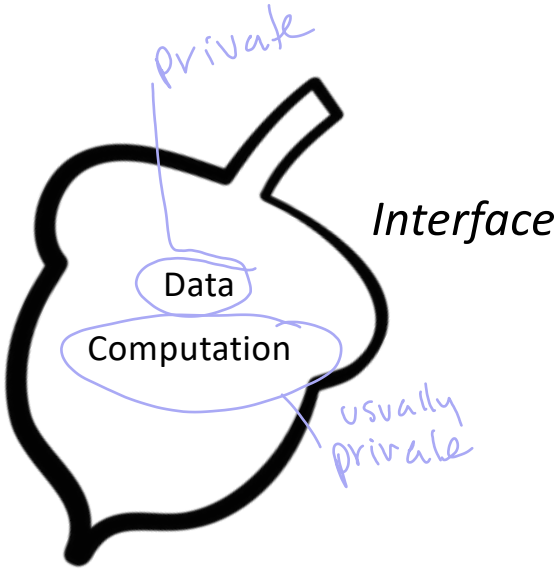
- Design techniques:

Interface-based behavior specification, UML Class Diagrams


Data
Data
Computation
Computation
Data
Computation
Data
Computation


Data
Computation
Data
Computation
Data
Computation
Data
Computation

Object Interaction



usually private

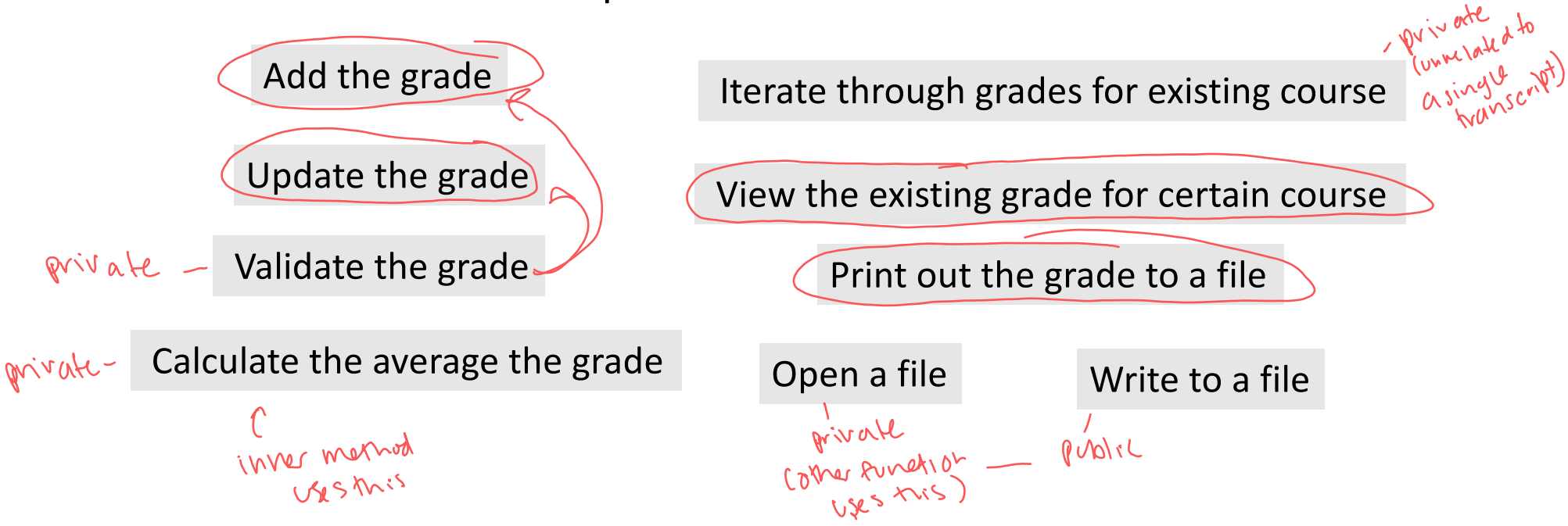
Supply the service through interface

interact through interface

Activity 1:

Which ones should be public?

- Thank about the design of a Class **StudentTranscipt**, which provides the basic functions related to the students' grade. What kind of methods should this class provide?



only make essential things public

Specification of public interface

- Requires — What needs be true in order call this the method?
- Modifies — When this method is called, is the state of any object going to be changed? (is internal state changing?)
- Effects — What will happen if this method is called?

Add the grade

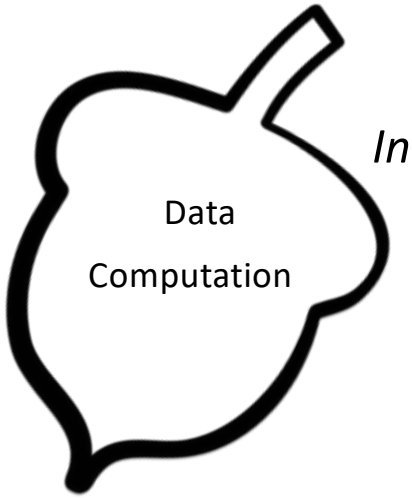
- required - grade, course, student
  valid grade
- modifies — transcript object
- effect - transcript reflects added grade

Java Interface Type

helps you specify a class interface before implementing

- Specification of related methods
- Reference to invoke those methods
- No implementation yet (except default and static methods)

```
public interface Student {
    /*
    * @return The unique id associated with student
    */
    String getID();
    /*
    * @return The first name of the student.
    */
    String getFirstName();
    /*
    * @return The Last name of the student.
    */
    String getLastName();

}
```



Interface

separate interface

Subtype Relationship

```
public class Undergrad implements Student
```

```
Student s1 = new Undergrad();

String id = s1.getID();
```

1. Undergrad need to provide implementation of methods in Student
2. Objects of Undergrad can be referred using variables of type Student.

Undergrad is-a **Student** (subtype relation)

all undergrads are students

not all students are undergrads

Why do we need this?

Student is interface type

```
public class Undergrad implements Student

public class Graduate implements Student

public class NonDegreeStudent implements Student

public class VisitingStudent implements Student
```

Program through interface

Polymorphic **Student**

Extensibility

Loosely coupling

Program to the interface

```
public boolean attendSeminar(Student pStudent)
{
     if(registeredStudents.size()<=cap) {
          registeredStudents.add(pStudent.getID());
          return true;

     }
     return false;
}
```

Polymorphism

- Many + Forms
- In programming languages, it's the ability to present the same interface for different underlying types.

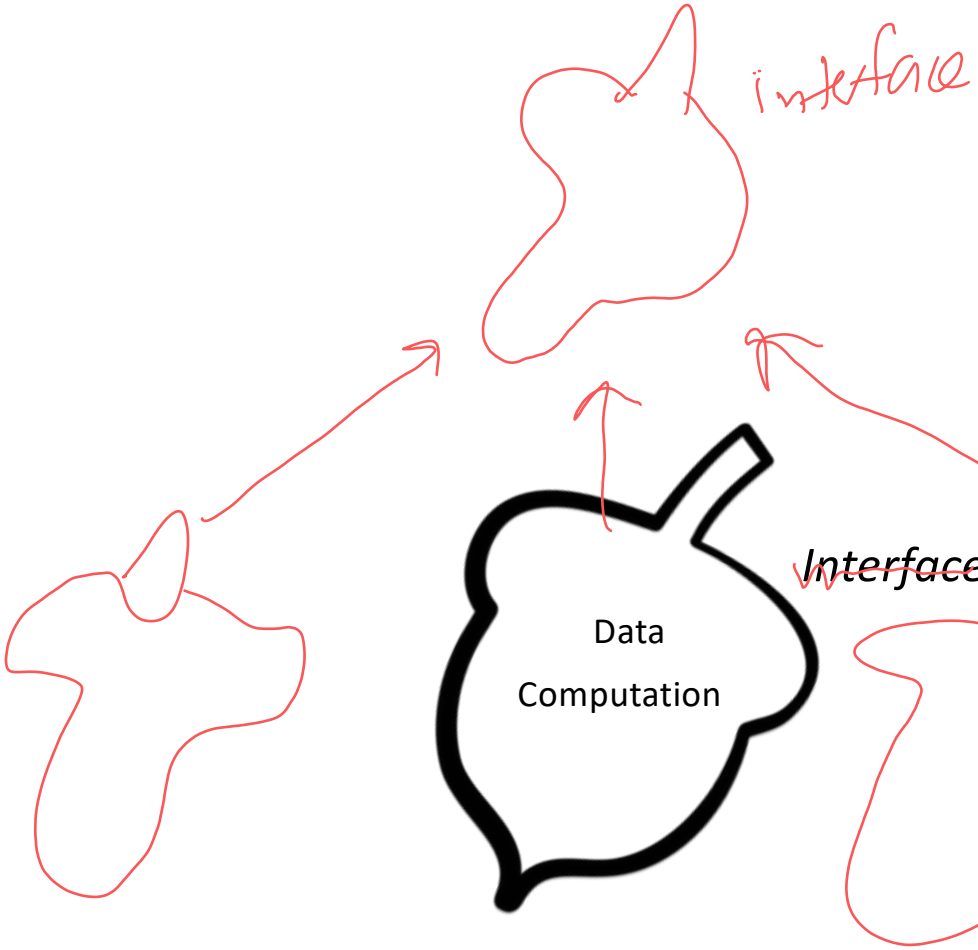
Image source: Griffith Ecology Lab (https://griffithecology.com/)

Comparison between Subtype and Subclass

- Subtype is about substitution:
  - B is a subtype of A means that if whenever the context requires an element of type A it can accept an element of type B.

- Subclass (one type of subtype) is about inheritance:
  - B is a subclass of A means that B can reuse unchanged fields and methods from A.
  - Extra dependencies between A and B
  - More in Later Modules (about inheritance)

```
public class SpecialDeck extends Deck {

        … …
}
```



What computation should be specified in the interface?

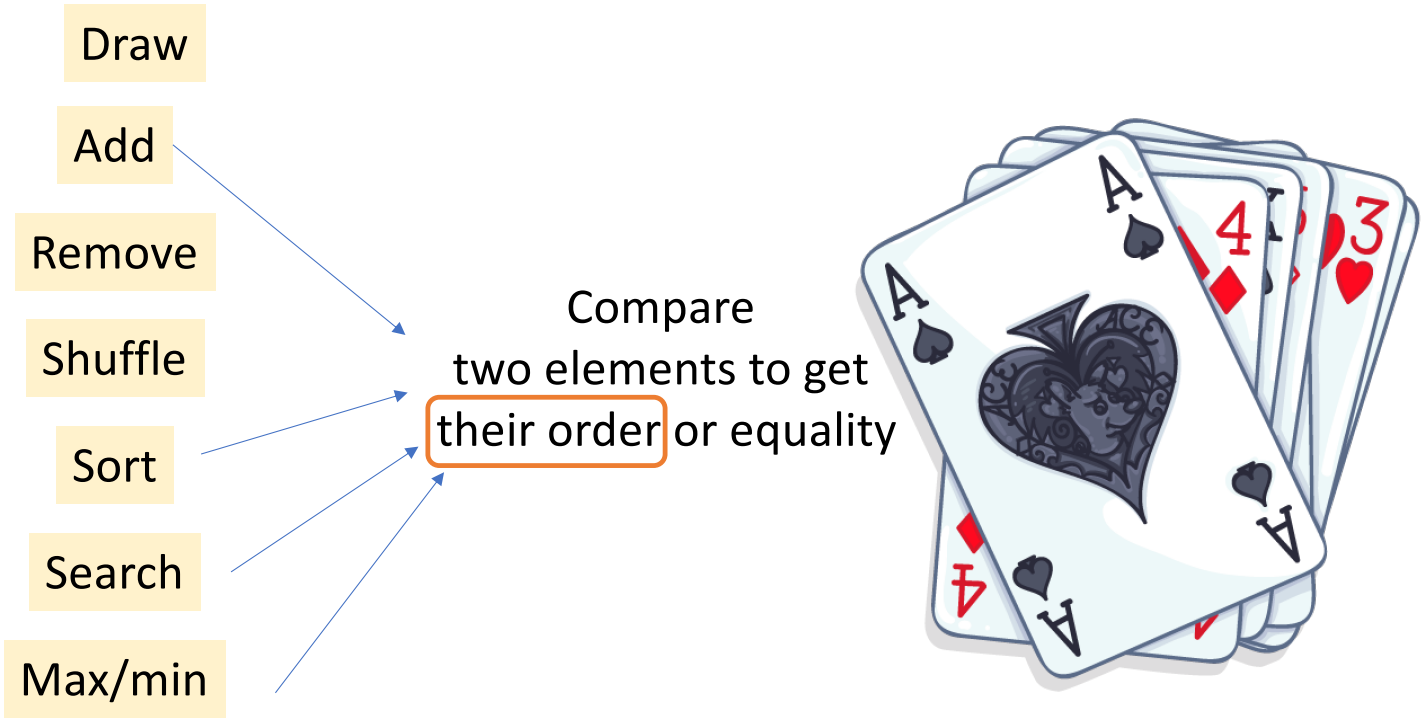
- public methods

Look for **Orthogonality**

↳ relevant things should change, irrelevant things should not

if all subtypes use a method, it should be in the interface

Operation on a Deck



Information leaking

a design knowledge is reflected in many modules

Java `Comparable<T>` Interface

- This interface imposes a total ordering on the objects of each class that implements it.

```
public interface Comparable<T>
{
        int compareTo(T o);
}
```

Generics: mechanism that takes type as parameter

needs body

Specification of `Comparable<T>`

- Compares this object with the specified object for order.

- Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

- Also properties of implementor needs to ensure, for example:

  (x.compareTo(y)>0 && y.compareTo(z)>0) implies x.compareTo(z)>0

Client

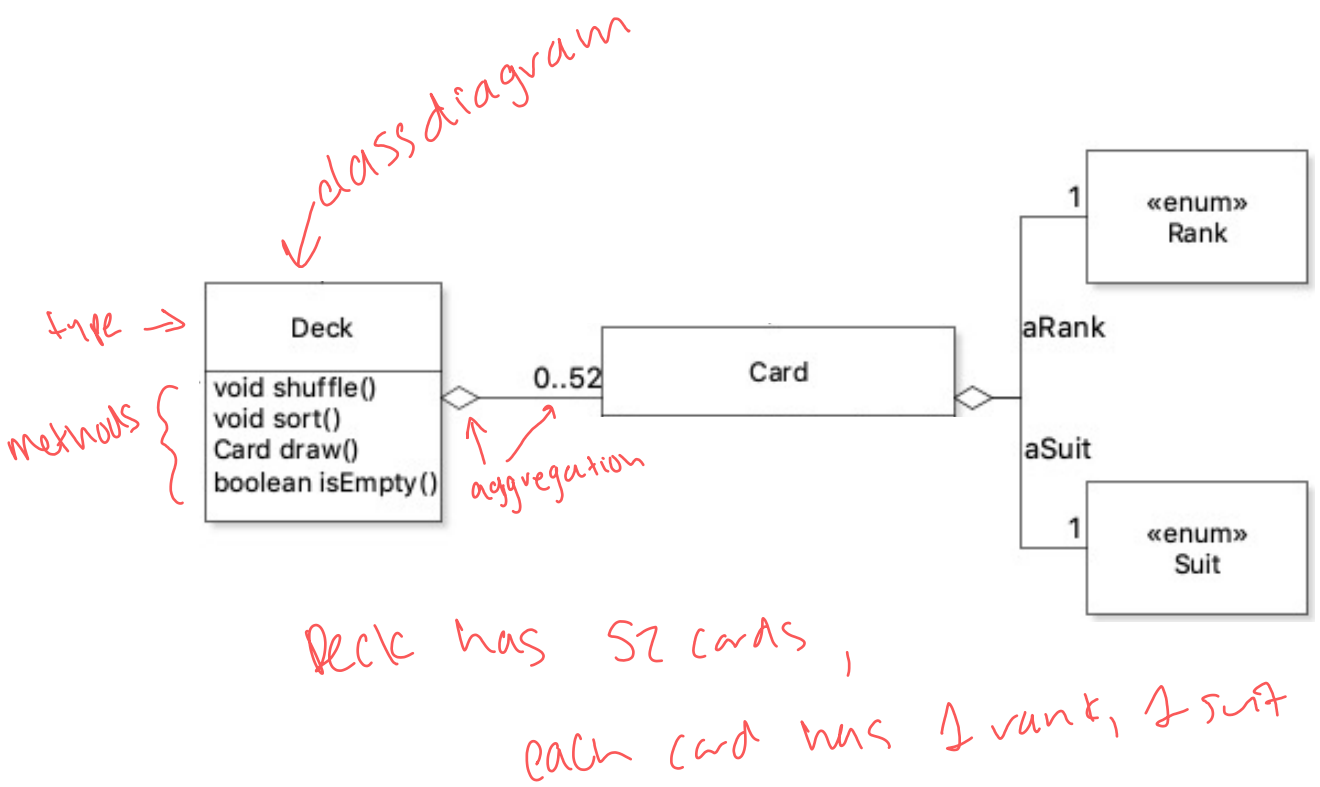
```
if(object1.compareTo(object2) >0) /*...*/
```

Implements `Comparable<T>`

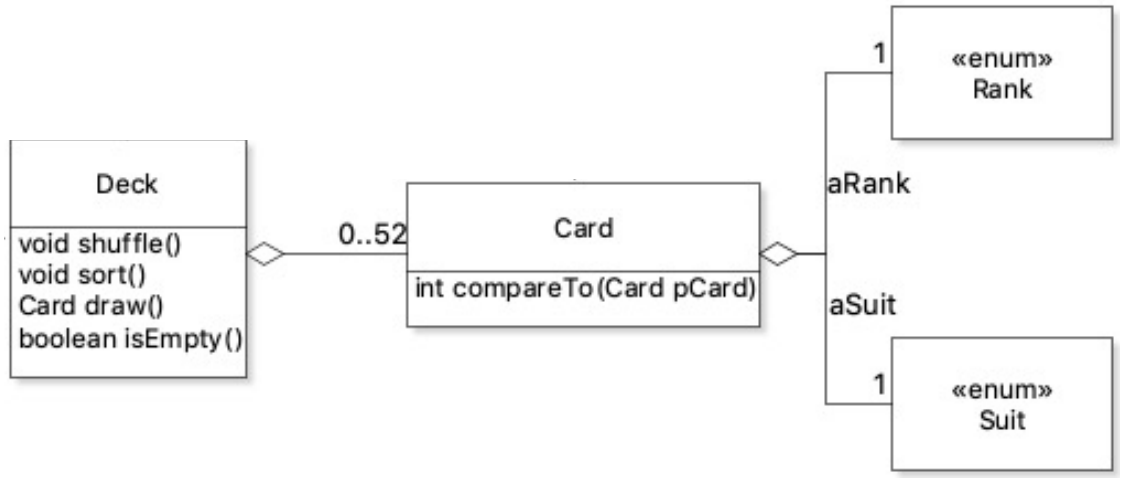
```
Collections.sort(aCards);// aCards is a List<Card> instance
```

```
public class Card implements Comparable<Card>
{

 … …


   @Override
   public int compareTo(Card pCard)
   {
   … …   return aRank.compareTo(pCard.aRank);
   }
}
```

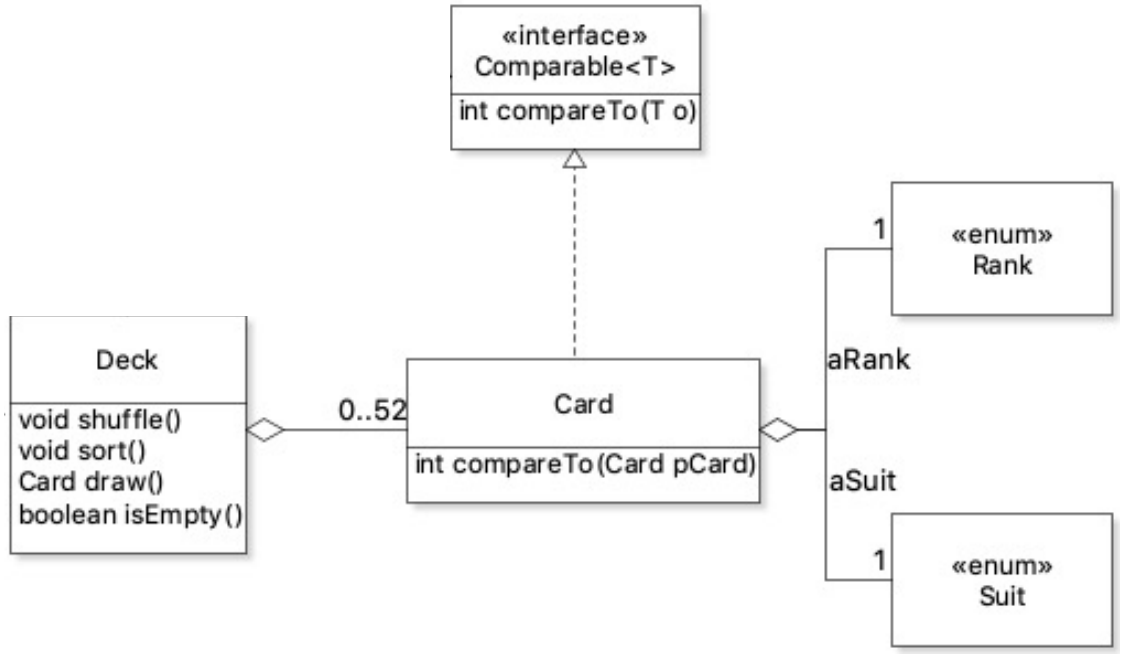
Current Design of Deck



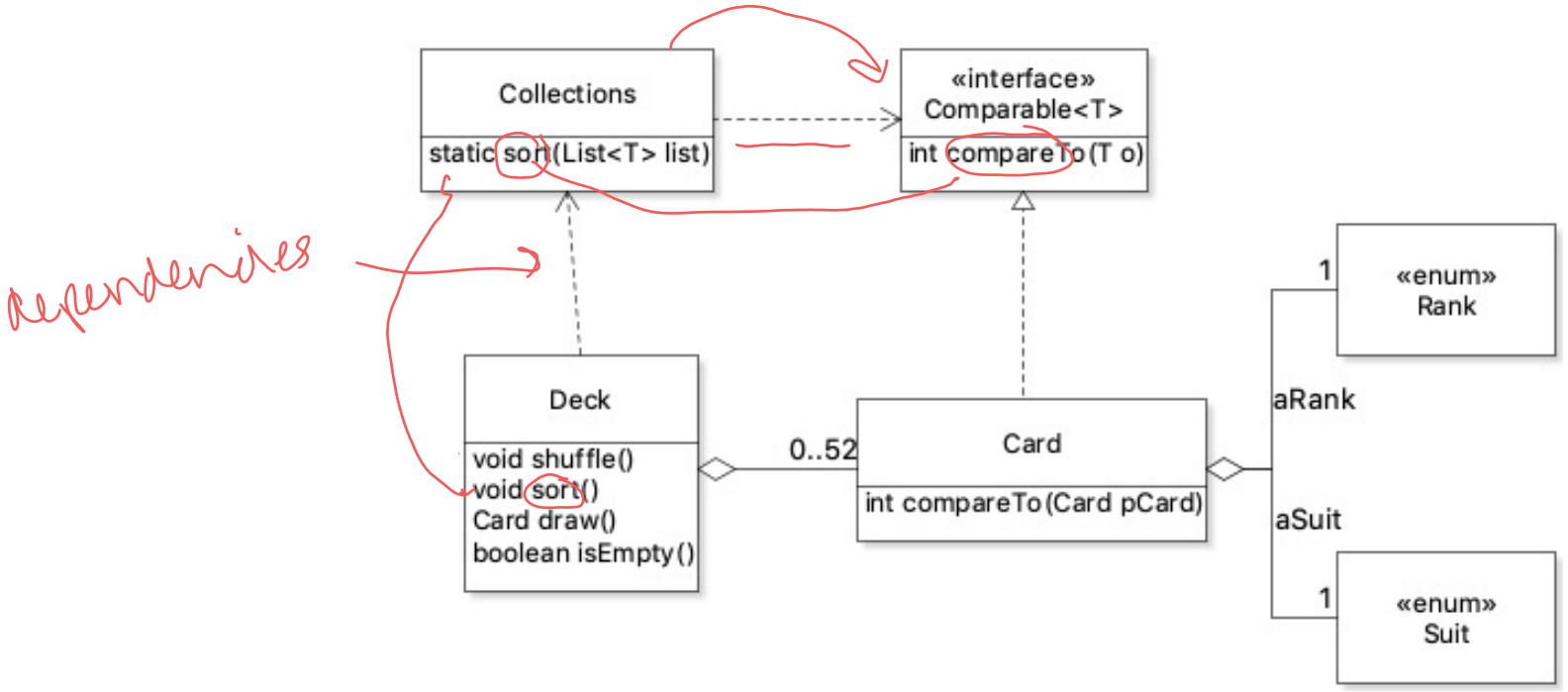
Current Design of Deck



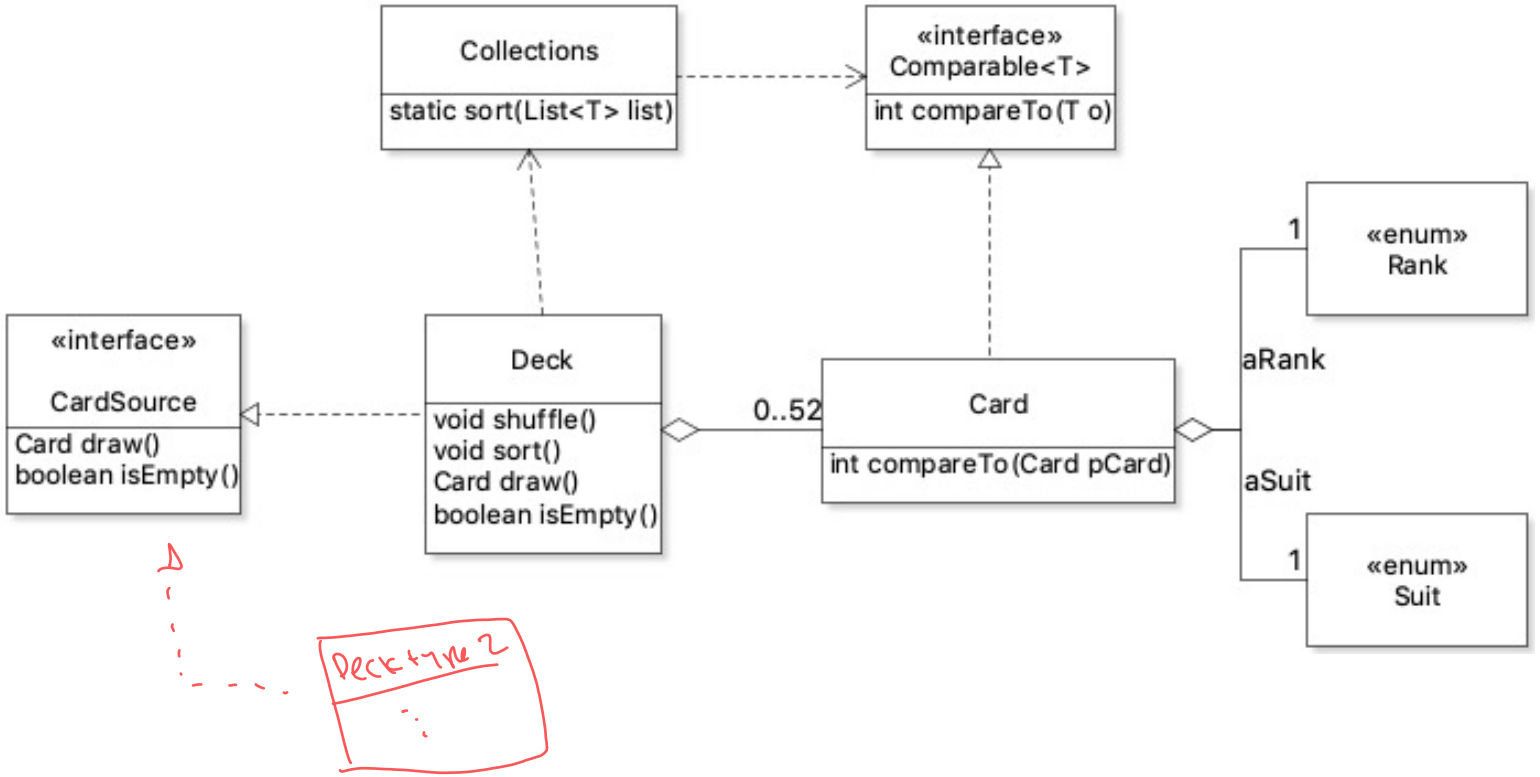
Current Design of Deck



Current Design of Deck

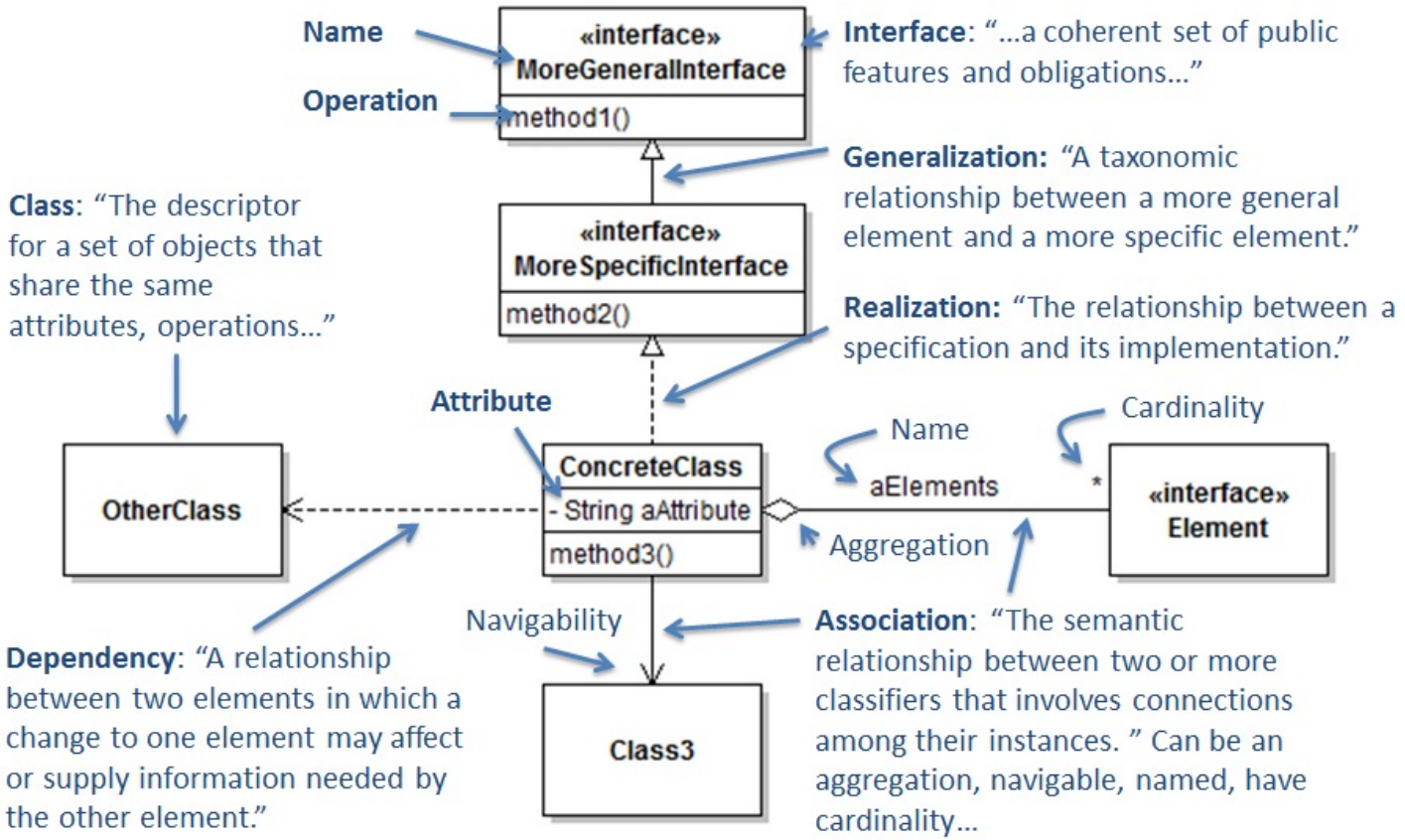


Current Design of Deck



UML Class Diagram

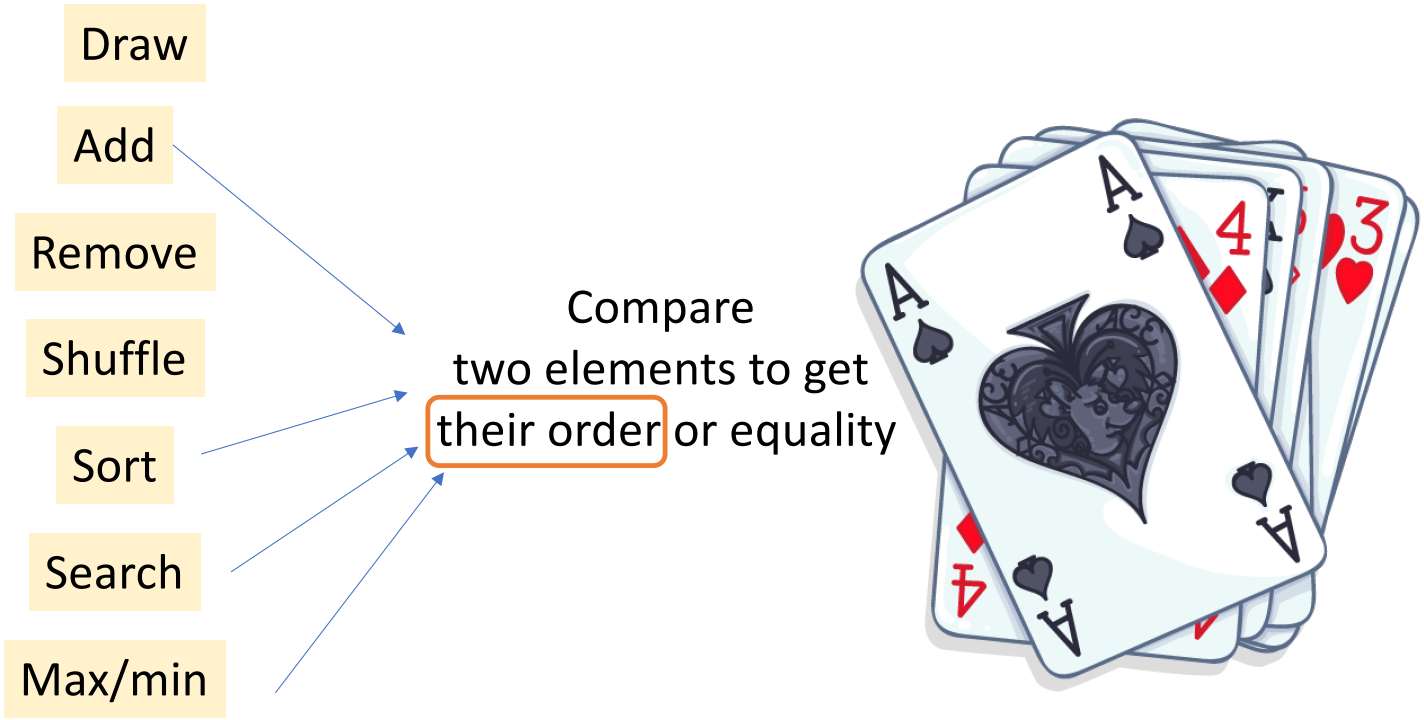
- Represent Type (mainly classes and interfaces) definitions and relations
- *Static* view (cannot show *run-time* properties)
- Tool: JetUML


Name
«interface»
MoreGeneralInterface
Operation
method1()
Interface: "...a coherent set of public features and obligations..."
Generalization: "A taxonomic relationship between a more general element and a more specific element."
«interface»
MoreSpecificInterface
method2()
Realization: "The relationship between a specification and its implementation."
Class: "The descriptor for a set of objects that share the same attributes, operations..."
Attribute
Name
Cardinality
ConcreteClass
aElements
*
«interface»
Element
OtherClass
- String aAttribute
method3()
Aggregation
Navigability
Association: "The semantic relationship between two or more classifiers that involves connections among their instances. " Can be an aggregation, navigable, named, have cardinality...
Dependency: "A relationship between two elements in which a change to one element may affect or supply information needed by the other element."
Class3

Separation of Concern

- Concern: anything that matters in providing a solution to a problem
- Prevent information Leakage
- To achieve “orthogonality”: changes in one does not affect any of the others.

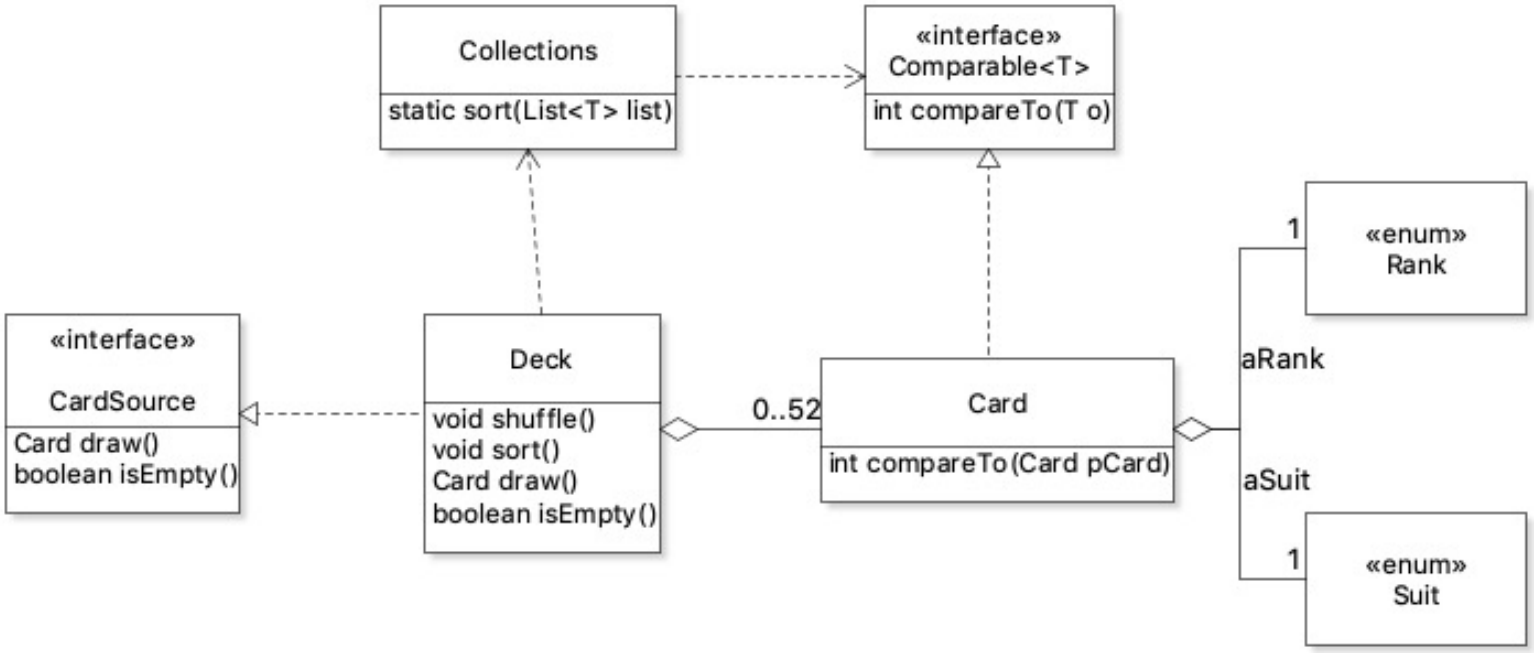
Operation on Card Collections



Information leaking

a design knowledge is reflected in many modules

How did this design apply the principle of Separation of Concern?



Summary

- Concepts and Principles:

Class's interface, Separation of concerns

- Programming mechanism:

Java Interface type, Subtype polymorphism

- Design techniques:

Interface-based behavior specification, UML Class Diagrams