



Jin L.C. Guo

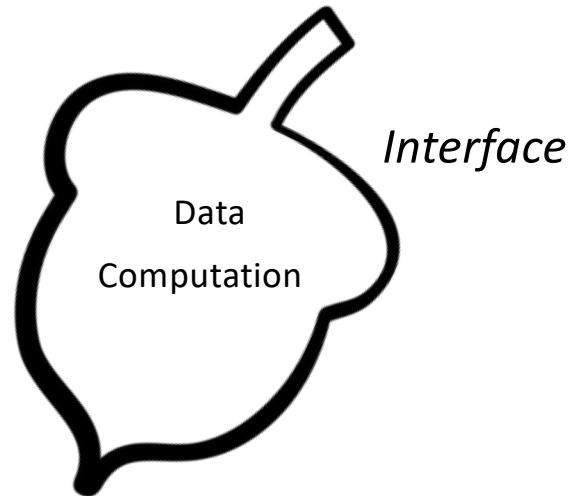
M2 (a) - Types and Polymorphism

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

Objective

- Interface type vs a class's interface;
- Subtype polymorphism
- Separation of concerns
- UML Class Diagrams

Object Interaction



Supply the service through interface

Activity 1:

- Evaluate the design of Class **Tokenizer**, which split input text into individual words (tokens), is it easy to be used or reused?

```
class Tokenizer {  
    public Tokenizer(InputStreamReader reader) {...}  
    public void readNextLine() throws IOException {...}  
    public int numTokens() {...}  
    public String getToken(int tokenPos) {...}  
}
```

Standard Deck of 52 Cards

```
public final class Deck
{
    private Stack<Card> aCards = new Stack<>();
    ...
    public Deck( Deck pDeck ) {...}

    public void shuffle() {...}
    public Card draw() {...}
    public boolean isEmpty() {...}
}

class Game
{
    Deck aDeck;
    ....
}
```

If we are building a library?

- User want to use the services to their own code for drawing cards
 - not 52 cards
 - don't need to be shuffled

isEmpty()
draw() are all they need!

Java Interface

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

```
public interface CardSource {  
    /*  
     * Remove a card from the source and return it.  
     *  
     * @return The card that was removed from the source  
     * @pre !isEmpty()  
     */  
    Card draw();  
    /*  
     * @return True if there is not card in the source.  
     */  
    boolean isEmpty();  
}
```

Subtype Relationship

```
public class Deck implements CardSource
```

```
CardSource source = new Deck();  
  
if(!source.isEmpty())  
    source.draw();
```

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

Deck is-a CardSource (subtype relation)

```
public class StandardDeck implements CardSource  
  
public class TwoDeck implements CardSource  
  
public class ClubDeck implements CardSource  
  
public class OrderedDeck implements CardSource
```

Polymorphic CardSource

Extensibility

Loosely coupling

Program to the interface

```
public static List<Card> drawCard(CardSource source, int pNumber)  
{  
    List<Card> result = new ArrayList<>();  
    for(int index = 0; index < pNumber && !source.isEmpty(); index++)  
    {  
        result.add(source.draw());  
    }  
    return result;  
}
```

Polymorphism

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



“Colour polymorphism is likely to be disadvantageous to some populations and species due to genetic architecture and morph interactions”, Griffith Ecology Lab (<https://griffithecology.com/>)

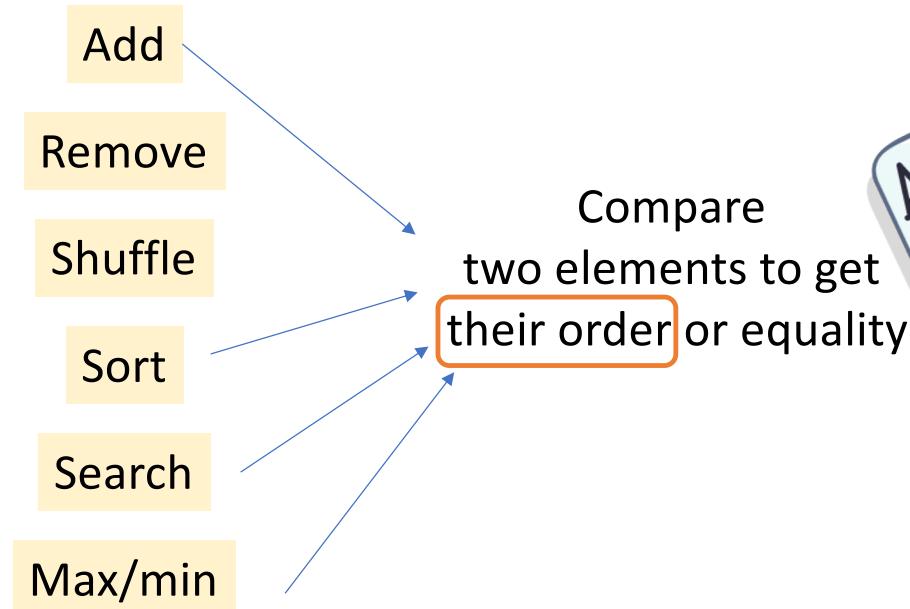
Specifying Behavior with Interface

- What to separate?
- What to specify?

Collection



Collections



Information leaking

a design knowledge is reflected in many modules

Java Comparable 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

Specification of Comparable

- 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:
 $(x.compareTo(y) > 0 \&& y.compareTo(z) > 0) \text{ implies } x.compareTo(z) > 0$

Client

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

Implements Comparable

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

```
public class Card implements Comparable<Card>
{
    ...
    @Override
    public int compareTo(Card o)
    {
        ... return aRank.compareTo(pCard.aRank);
    }
}
```

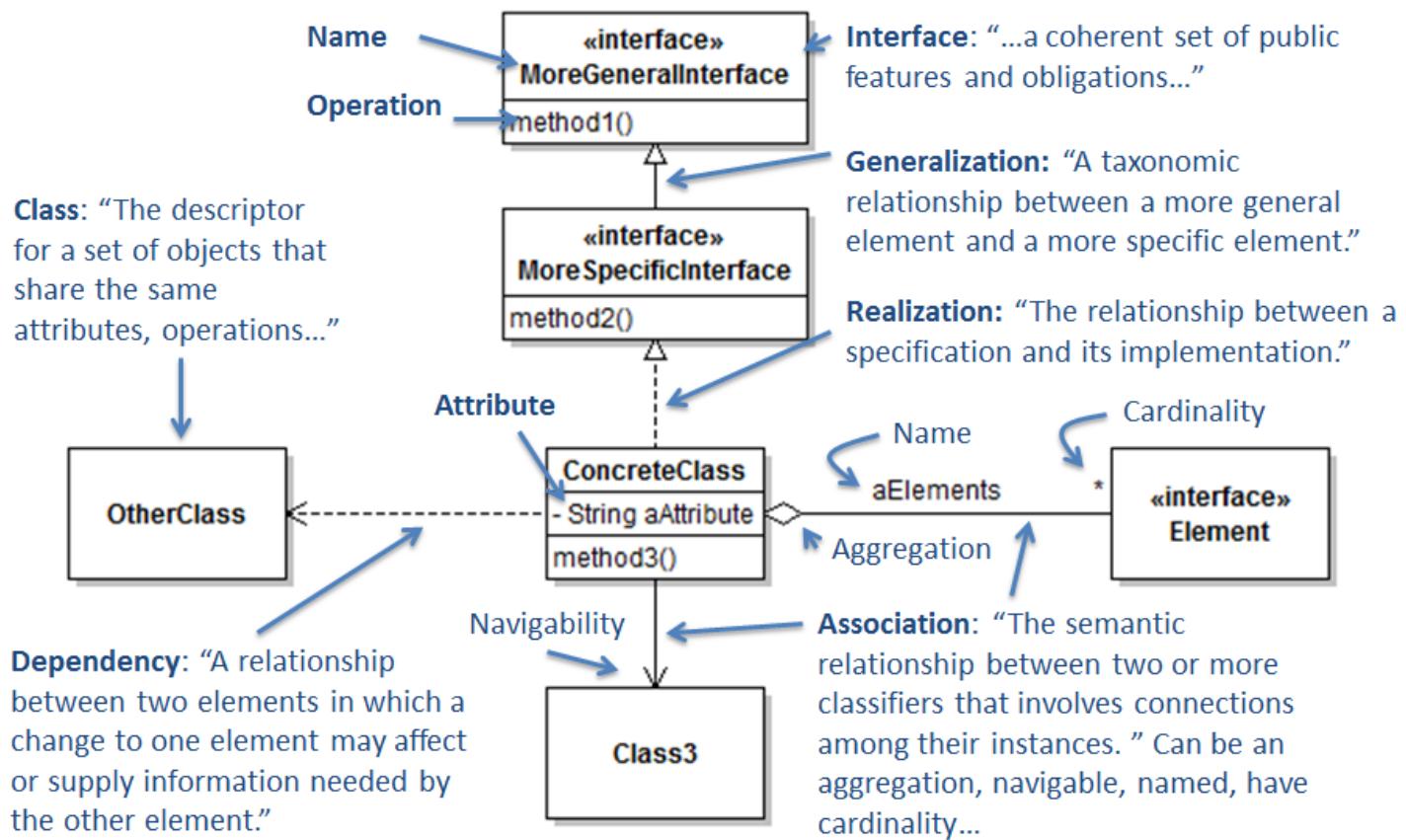
The base class of all Java language enumeration types has implemented Comparable<E>

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.

UML Class Diagram

- Represent Type (mainly classes and interfaces) definitions and relations
- *Static* view (cannot show *run-time* properties)
- Use the tool JetUML for this class.



Activity 2: Class Diagram for Deck

- Draw the class diagram for a standard deck class including functions of shuffling, drawing, and sorting.

Class Diagram for Deck

