Barbara Liskov

- American
- MIT
- 2008 Alan Turin award
- The Liskov principle

(B. Liskov, J. Wing, 1994)

Extension and Contraction

- Inheritance can **extend**:
  - the behavior and data associated with the child class is a larger set that the behavior and data associated with the parent class
- Inheritance can **contract**:
  - the child class is a more specialized or restricted form of the parent class

Example (Contraction)

```
public class CardPile {
        int x, y;
        Stack thePile;
        public CardPile (int x1, int yl) {
           x = xl;
           y = yl;
           thePile = new Stack();
        }
        public void addCard (Card aCard) {
             thePile.push(aCard);
        }
```

```
class DiscardPile extends CardPile {
        public DiscardPile (int x, int y) {
              super (x, y);
        }
        public void addCard (Card aCard){
              if (!aCard.faceUp()){
                   aCard.flip();
                   super.addCard(aCard);
              }
        }
        ...
}
```

The addCard method has been restricted

Example (Extension)

```
public class CardPile {
     int x, y;
     Stack thePile;
     public CardPile (int x1, int yl) {
          x = xl;
          y = yl;
          thePile = new Stack();
     }
     public void addCard (Card aCard) {
          thePile.push(aCard);
     }
```

```
     Suite mySuite;
     public int countSuiteCards(){
          return mySuite.length;
     }
}
```

A new method is added

Subclass, Subtype and Substitutability

- Subclassing and subtyping is not exactly the same thing

Liskov principle of substitutability

- The **Liskov** principle of substitutability says that given two classes A and B, where **B is a subclass of A,** it should be possible

  to replace any instance of class A with instances of class B in any situation with no observable effect

Example

- **RacingBike** and **LightedBike** classes are subclasses of **Bike**
- **Bike** class has **goHome()** method that
  - **Pre-condition**: Requires that the **sun has not yet set** and
  - **Post-condition**: it gets you home in **exactly 10 minutes**

Example

- In **LightedBike**, goHome() has
  - **Weaker Pre-condition**: It does not require that the sun has not yet set, since it has a light it will work perfectly well under this adverse condition
  - **Same Post-condition**: it gets you home in 10 minutes,
- Then **LightedBike satisfies the contract of Substitutability** because it requires less and promises the same effects

Pre-conditions

- A pre-condition tells you when to use the class
- The pre-condition is typically seen in the code in which the instance is used:

```
if (!Sun.set(t)){
      Bike myBike= new Bike();
      myBike.goHome();
}
```


Substituting

```
if (!Sun.set(t)){
      Bike myBike= new LightedBike();
      myBike.goHome();
}
```

Liskov principle of substitutability

- If **RacingBike**, goHome() has
  - **Same Pre-condition**: requires that the sun has not yet set,
  - **Post-condition changing the invariant:** (time to go home): gets you home in **5 minutes**,
- Then RacingBike **breaks** the contract of Substitutability

Liskov principle of substitutability

- Even though someone might consider it beneficial that the method takes 5 minutes less to perform, Bike specifies that it would take exactly 10 minutes
- If a **code relies** on the fact that the **goHome()** would take **10 minutes** and someone has maliciously passed a RacingBike when it was expected a Bike, the **code would no longer behave predictably**

Liskov principle of substitutability

- The **base cannot be extended without to code it again**
  - If I wanted to extend Bike with RacingBike then I had to re-code Bike in order it to know about the behaviour of RacingBike
  - In our case, Bike has to allow going home in 5 minutes too (e.g, between 10 and 5 minutes)

Open/closed principle

- Breaking the substitutability principle implies breaking the **open/closed principle** of object programming

Open/closed principle

Open/closed principle Bertrand Meyer (1988 "Object Oriented Software Construction" book):

an entity is open to extension and closed to implementation

- Subclasses must extend bases without any further implementation of the code of the base

Liskov conditions

- Preconditions **cannot** be **strengthened** in a subtype
- Postconditions **cannot** be **weakened** in a subtype
- **Invariants** of the supertype must be **preserved** in a subtype

Subtyping and Subclassing

- Subtyping and subclassing are different in how they do or do not follow the Liskov principle.
  - Subtyping **does follow** the principle of Liskov

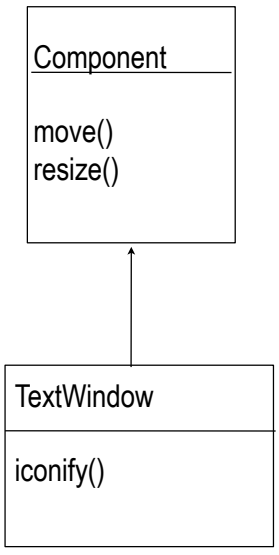
Forms of Inheritance

- Inheritance is used in a surprisingly variety of ways. Some general abstract categories are:
  - Subclassing for specialization
  - Subclassing for specification
  - Subclassing for construction
  - Subclassing for generalization
  - Subclassing for extension
  - Subclassing for limitation
  - Subclassing for variance
  - Subclassing for combination

Forms of Inheritance

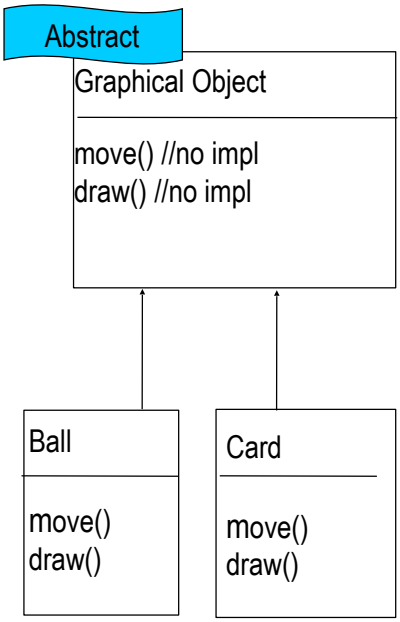
- The different types of inheritance depend on how the specifications of the parent classes are respected or modified in the child classes

Subclassing for Specialization



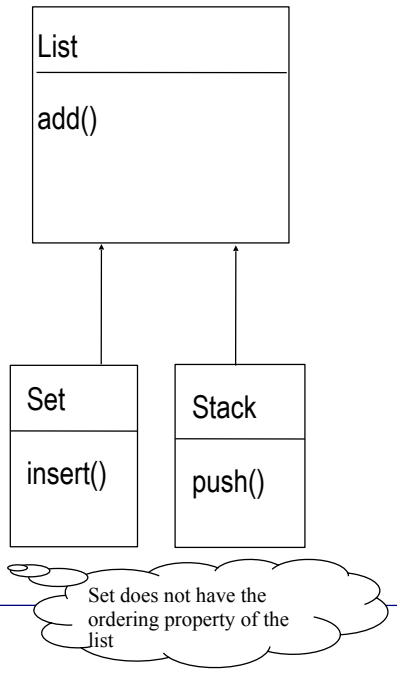
- The child class is a specialized form of the parent class but satisfies the specifications of the parent class completely
- **The principle of substitutability is explicitly upheld**
- Is an ideal form of inheritance, good designs should strive for it

Subclassing for Specification



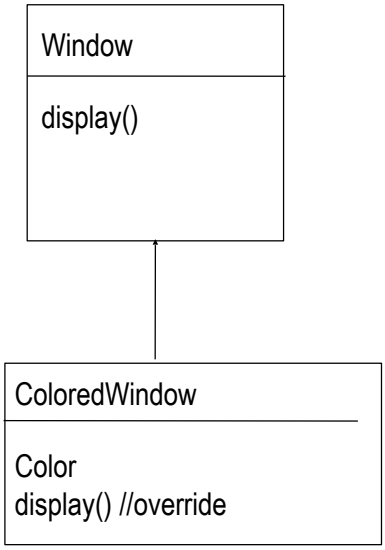
- The child class implements the behaviour described, which it is not implemented in the parent class though
- The parent class is an abstract class; it is not permitted to create instances of it
- **It does not break the Liskov principle** as children implement exactly the specifications of the parent class

Subclassing for Construction



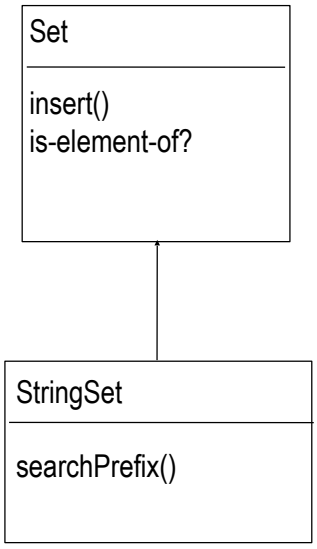
- The child class gets most of its desired functionality from the parent class only changing names of methods or modifying arguments
- But the methods of the parent **are not part of the specifications** of the subclass
- **It breaks the principle of substitutability intentionally**
- Opens a fast and easy route to new data abstractions

Subclassing for Generalization



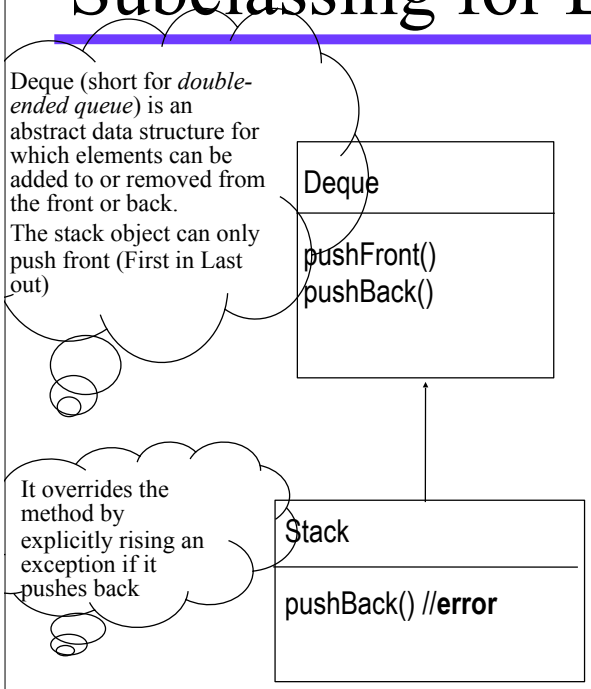
- The child class modifies or extends the parent class to obtain a more general kind of object. A colored window generalizes a window class that might have only white background
- At least one method of the parent must be overridden.
- Used when building upon a base of existing classes that are difficult to modify
- Should be **avoided** in favour of inverting the class hierarchy and using subclassing for specialization
- **It is not subtyping**

Subclassing for Extension



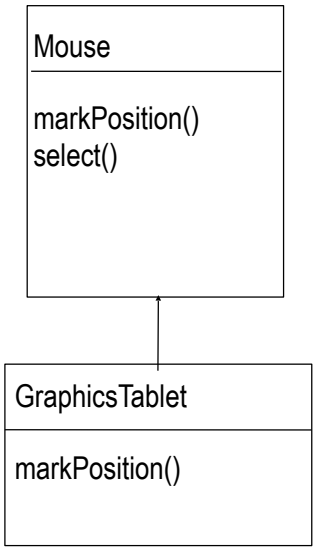
- Adds totally new abilities to the subclass;
- new methods **are added** to those of the parent with a functionality that **is not strongly tied** with the existing methods
- As the functionality of the parent class remains untouched **it obeys the principle of substitutability**

Subclassing for Limitation



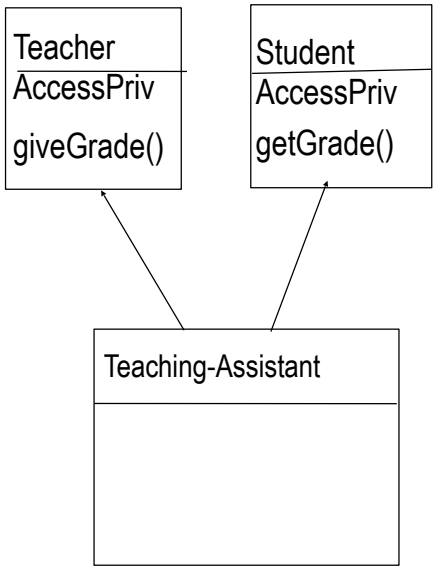
- The subclass modifies or overrides methods of the parent class to eliminate functionality leaving a subclass with smaller or more restrictive behavior
- **In an explicit contravention of the principle of substitutability**
- Also used when an existing base is hard to modify

Subclassing for Variance



- It is employed when two classes have similar implementations but no conceptual hierarchical relation
- One of the two classes is arbitrarily selected to be parent; the common code in inherited, the specific code is overridden
- Usually a better alternative is to factor out the common code and have the two classes inherit from a common superclass
- **It breaks the substitutability principle**

Subclassing for Combination



- Used to give the subclass a combination of features from two or more parent classes
- Also known as multiple inheritance; in java implemented with delegation
- Becomes a problem when clashes occur
- **It does follow the Liskov principle**