Interfaces
Dynamic Binding

Java Interfaces

- It defines a standard and public way of specifying the behavior of classes
- All methods of an interface are abstract methods
  - Defines the signatures of a set of methods, without the body (implementation of the methods)
- A concrete class must implement the interface (all the abstract methods of the Interface)
- It allows classes, regardless of their locations in the class hierarchy, to implement common behaviors

```
public interface Relation {

    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
    public boolean isEqual( Object a, Object b);

}
```

```
public class Line implements Relation {
    private double x1;
    private double x2;
    private double y1;
    private double y2;

public Line(double x1, double x2, double y1, double y2){
        this.x1 = x1;
        this.x2 = x2;
        this.y1 = y1;
        this.y2 = y2; }
```

```
public double getLength(){
    double length = Math.sqrt((x2-x1)*(x2-x1) +
    (y2-y1)* (y2-y1));
    return length; }

public boolean isGreater( Object a, Object b){
    double aLen = ((Line)a).getLength();
    double bLen = ((Line)b).getLength();
    return (aLen > bLen); }

public boolean isLess( Object a, Object b){
    double aLen = ((Line)a).getLength();
    double bLen = ((Line)b).getLength();
    return (aLen < bLen);}

public boolean isEqual( Object a, Object b){
    double aLen = ((Line)a).getLength();
    double bLen = ((Line)b).getLength();
    return (aLen == bLen); }  }
```

All data fields are *public final static* and all methods are *public abstract* in an interface. For this reason, these modifiers can be omitted, as shown below:

```
public interface T1 {
  public static final int K = 1;

  public abstract void p();
}
```

Equivalent

```
public interface T1 {
  int K = 1;

  void p();
}
```

A constant defined in an interface can be accessed using syntax InterfaceName.CONSTANT_NAME (e.g., T1.K).

- The `final` class cannot be extended:

```
final class Math {
  ...
 }
```

- The `final` variable is a constant:

```
final static double PI = 3.14159;
```

- The `final` method cannot be overridden by its subclasses.

The modifiers are used on classes and class members (data and methods), except that the final modifier can also be used on local variables in a method. A final local variable is a constant inside a method.

- To reveal an object's programming interface (functionality of the object) without revealing its implementation
  - This is the concept of encapsulation
  - The implementation can change without affecting the caller of the interface
  - The caller does not need the implementation at the compile time
    - It needs only the interface at the compile time
    - During runtime, actual object instance is associated with the interface type

- To have unrelated classes implement similar methods (behaviors)
  - One class is not a sub-class of another
- Example:
  - Class Line and class MyInteger
  - They are not related through inheritance
  - You want both to implement comparison methods
    - checkIsGreater(Object x, Object y)
    - checkIsLess(Object x, Object y)
    - checkIsEqual(Object x, Object y)
  - Define Comparison interface which has the three abstract methods above

- To model multiple inheritance
  - A class can implement multiple interfaces while it can extend only one class

- All methods of an Interface are abstract methods while some methods of an Abstract class are abstract methods
  - Abstract methods of abstract class have abstract modifier
- An interface can only define constants while abstract class can have fields
- Interfaces have no direct inherited relationship with any particular class, they are defined independently

- Interfaces themselves have inheritance relationship among themselves

- If you define a reference variable whose type is an interface, any object you assign to it must be an instance of a class that implements the interface
- Let's say Person class implements PersonInterface interface
  - Person p1 = new Person();
  - PersonInterface pi1 = p1;
  - PersonInterface pi2 = new Person();

```
public class ComputerScienceStudent extends Student implements
    PersonInterface, AnotherInterface, Third interface
    {
        // All abstract methods of all interfaces
        // need to be implemented.
    }
```

An interface can only be implemented by classes or extended by other interfaces

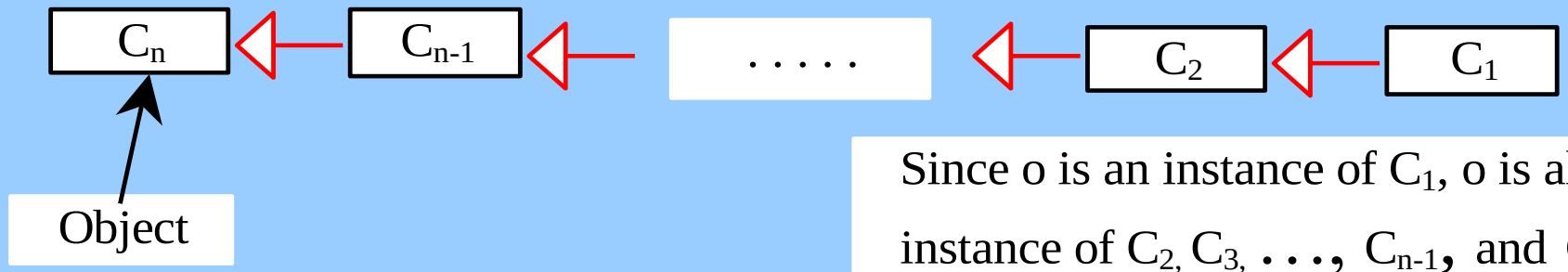
- Interfaces are not part of the class hierarchy. However, interfaces can have inheritance relationship among themselves

```
public interface PersonInterface {
    void doSomething();}

public interface StudentInterface extends
    PersonInterface {
    void doExtraSomething();}
```

Dynamic binding works as follows: Suppose an object o is an instance of classes C_1, C_2, ..., C_{n-1}, and C_n, where C_1 is a subclass of C_2, C_2 is a subclass of C_3, ..., and C_{n-1} is a subclass of C_n. That is, C_n is the most general class, and C_1 is the most specific class.

In Java, C_n is the Object class. If o invokes a method p, the JVM searches the implementation for the method p in C_1, C_2, ..., C_{n-1} and C_n, in this order, until it is found. Once an implementation is found, the search stops and the first-found implementation is invoked.



- Reading Material:
  - Chapter 10: 395-419