Java Allows only single parent
- i.e. lets classes have only one superclass
- Because multiple inheritance can cause problems
- E.g. Consider a Person class with resolveConflict() method

Problems with Multiple Inheritance
- Governor might override resolveConflict() to mediate a solution
- Terminator might override resolveConflict() a slightly differently
- What is a Governor to do?
  - Which superclass resolveConflict() method does it inherit
- This known as Diamond Inheritance
Interface

- To avoid diamond inheritance Java does not allow extending more than one class in Java.

- Instead, we use an interface which allows us to specify behavior that we want to have without actually specifying the exact implementation.

```java
public interface Trainable{
    public void sit();
    public void stand();
}
```

An interface is like a contract, protocol, role, or point of view.

- Code written for an interface type works with any object whose class implements it.
- It can assume that all of the subtypes have the methods listed in the interface (e.g. sit and stand).

A Class implements an Interface

```java
public class Dog implements Trainable{
    public void sit(){ // code for sit method }
    public void stand(){ // code for stand method }
}
```

A class that implements an interface

- Must provide concrete methods for each interface method.
- May have additional methods.
- May implement multiple interfaces.

```java
public class Dog implements Trainable, Comparable{
    // code
}
```
Example: Trainable Interface

> Lion lion = new Lion();
> Dog dog = new Dog();
> Trainable beast; // variable of type Trainable can refer to any object that implements the Trainable interface
> beast = lion; // beast is of type Trainable pointing to lion object
> beast.sit(); // calls lion's sit method
> beast = dog;
> beast.sit(); // calls dog's sit method

Abstract class vs. Interface

- When a partial implementation is feasible, abstract classes make sense as they can provide some functionality with the methods
- Pure abstract classes (with all abstract methods) in Java are functionally equivalent to an interface, but restricted to single inheritance
- Java will allow you to implement more than one interface
  - You can extend only one class but implement many interfaces
- You can use access modifiers (e.g., protected) in an abstract class though. Interfaces are always public.
- Also Interfaces can only have constants (implicitly public, static, and final)

Designing Interfaces

- Most of the time, you will use Sun-supplied Java interfaces
  - E.g. ActionerListner & KeyListerner for GUIs (Graphical User Interface)
- Sometimes you will want to design your own
- You would write an interface if you want classes of various types to all have a certain set of capabilities

Another Interface Example

public interface RuleSet {
    boolean isLegal(Move m, Board b);
    void makeMove(Move m);
}

- Every class that implements RuleSet must have these methods
- class CheckersRules implements RuleSet {
    // one implementation
    public boolean isLegal(Move m, Board b) { ... }
    public void makeMove(Move m) { ... }
}
- class ChessRules implements RuleSet { ... } // another implementation
- class LinesOfActionRules implements RuleSet { ... } // and another

    RuleSet rulesOfThisGame = new ChessRules();
    // This assignment is legal because a rulesOfThisGame object is a RuleSet object
    if (rulesOfThisGame.isLegal(m, b)) { makeMove(m); }
    // This statement is legal because, whatever kind of RuleSet object rulesOfThisGame is, it must have isLegal and makeMove methods
Extending and Implementing

- Class can only extend one other class
- But can implement multiple interfaces
- The exact order must be used as shown below in declaring a class below:

```java
public class SubclassName extends SuperclassName implements Interface1, Interface2 {
}
```

- Otherwise there is compile error

<table>
<thead>
<tr>
<th></th>
<th>Regular Class</th>
<th>Abstract Class</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
<td>all concrete</td>
<td>concrete and/or abstract</td>
<td>all abstract</td>
</tr>
<tr>
<td>May have instance variables</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>May be instantiated</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>