Code Reuse Through Hierarchies

Part 3: you will learn how interfaces provide a design guide while abstract classes can specify design (abstract methods) but also an common implementation (non-abstract methods)

James Tam

Java Interfaces

```
public interface BankAccount
{
    public final static int MIN_BALANCE = 0;
    public void displayBalance();
    public void deposit(double amount);
    public void withdraw(double amount);
}
```

- Similar to a class
- Provides a design guide rather than implementation details
- Specifies what methods should be implemented but not how (i.e. method signature but not method body)
  - (Specify the signature of methods so each part of the project can proceed with minimal coupling between classes).
  - An important design tool: Agreement for the interfaces should occur very early before program code has been written.
  - Changing the method body rather than the method signature won't 'break' code.
- It's a design tool so interfaces cannot be instantiated
  - Q: What if one could instantiate an interface directly?
- **Name of the folder containing the complete online example**: `8hierarchiesInterfaces`

James Tam

Interfaces: Format

Note the file defining a Java interface must end in dot-java e.g. `BankAccount.java`

Format for defining an interface

```
public interface <name of interface>
{
     constants
     methods to be implemented by the class that realizes
      (provides a body) this interface
}
```

Example for defining an interface

```
public interface BankAccount
{
    public final static int MIN_BALANCE = 0;
    public void displayBalance();
}
```

James Tam

Classes That Implement/Realize An Interface

Format for realizing / implementing the interface

```
public class <name of class> implements <name of interface>
{
     attributes
     methods actually implemented by this class
}
```

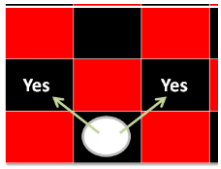
Format for realizing / implementing the interface

```
public class SavingsAccount
{
    private double balance;
    public void displayBalance()
    {
        System.out.println(balance);
    }
}
```

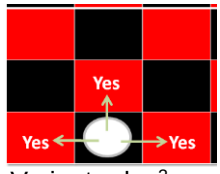
James Tam

Interfaces: A Checkers Example

Regular board[1]



Regular rules[2]



Variant rules[2]

1 From www.allaboutfungames.com)

2 Board images from "Tam"

James Tam

Interface Board

```
public interface Board
{
    public static final int SIZE = 8;
    public void displayBoard();
    public void initializeBoard();
    public void movePiece();
    boolean moveValid(int xSource,
                      int ySource,
                      int xDestination,
                      int yDestination);
        ...
}
```

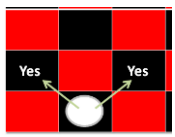
James Tam

Class RegularBoard

```
public class RegularBoard implements Board
{
    public void displayBoard()
    {
        ...
    }

    public void initializeBoard()
    {
        ...
    }
```


Class RegularBoard (2)

```
    public void movePiece() {
        //Get (x, y) coordinates for the source and destination
        if (moveValid(xS, yS, xD, yD) == true)
          //Actually move the piece
        else
          //Don't move piece and display error message
    }

    public boolean moveValid(int xSource, int ySource,
                             int xDestination,
                             int yDestination)
    {
        if (moving forward diagonally)
            return(true);
        else
            return(false);
    }
} // End of class RegularBoard
```



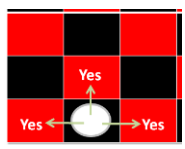
Class VariantBoard

```
public class VariantBoard implements Board
{
    public void displayBoard ()
    {
        ...
    }

    public void initializeBoard ()
    {
        ...
    }
```

James Tam

Class VariantBoard (2)

```
    public void movePiece() {
       //Get (x, y) coordinates for the source and destination
       if (moveValid (xS, yS, xD, yD) == true)
           // Actually move the piece
       else
           //Don't move piece and display error message
    }

    public boolean moveValid(int xSource, int ySource,
                             int xDestination,
                             int yDestination)
    {
       if (moving straight-forward or straight side-ways)
            return(true);
       else
            return(false);
    }
}  //End of class VariantBoard
```



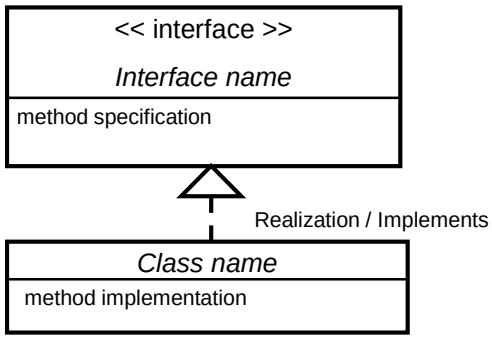
James Tam

Interfaces: Recapping The Example

- Interface Board
  - No state (variable data) or behavior (body of the method is empty)
  - Specifies the behaviors that a board *should* exhibit e.g., clear screen
  - This is done by listing the methods that must be implemented by classes that implement the interface.
- Class RegularBoard and VariantBoard
  - Can have state and methods
  - They must implement all the methods specified by the interface 'Board' (but can also implement other methods too)

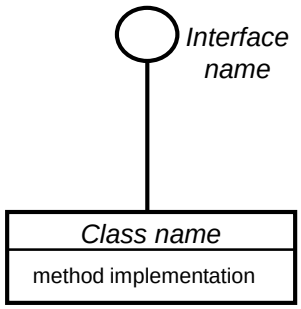
James Tam

Specifying Interfaces In UML



James Tam

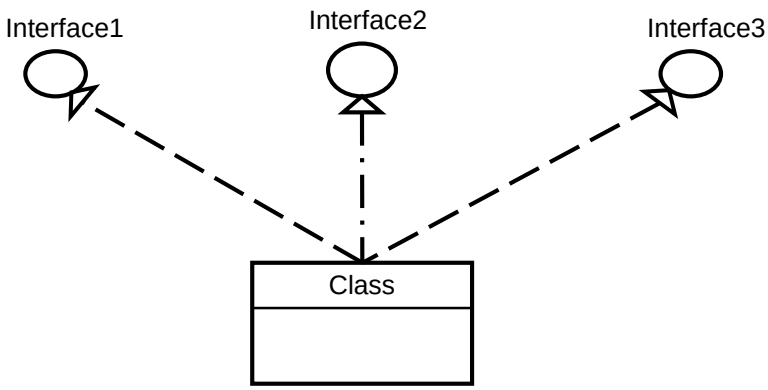
Alternate UML Representation (Lollipop Notation)



James Tam

Implementing Multiple Interfaces

- Java allows for this.



James Tam

Implementing Multiple Interfaces

Format:

```
public class <class name> implements <interface name 1>,
             <interface name 2>, <interface name 3>…
{

}
```

James Tam

Multiple Implementations Vs. Multiple Inheritance

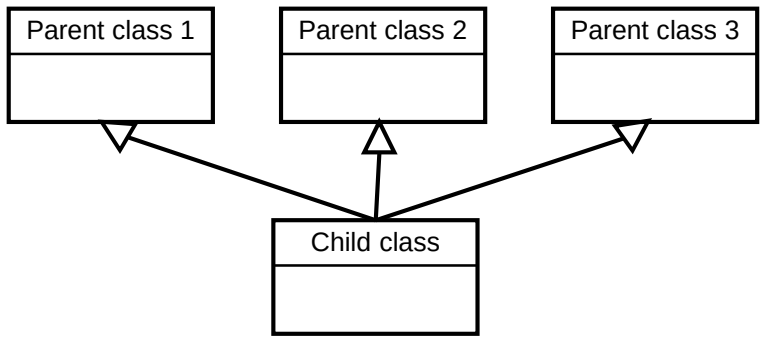
- A class can implement multiple interfaces
- Classes in Java cannot extend more than one class
- Again: multiple inheritance is **not possible in Java** but is possible in other languages such as C++:
  - Multiple inheritance (mock up code)

```
class <class name 1> extends <class
name 2>, <class name 3>…
{


}
```

James Tam

Multiple Implementations Vs. Multiple Inheritance (2)

- Multiple inheritance: conceptual view representing using UML



James Tam

Abstract Classes (Java)

- Classes that cannot be instantiated.
- A hybrid between regular classes and interfaces. Some methods may be implemented while others are only specified (no body).
- Used when the parent class:
  - specifies a default implementation of some methods,
  - but cannot define a complete default implementation of other methods (implementation must be specified by the child class).

- **Format:**

```
public abstract class <class name>
{
    <public/private/protected> abstract method ();
}
```

James Tam

Abstract Classes (Java): 2

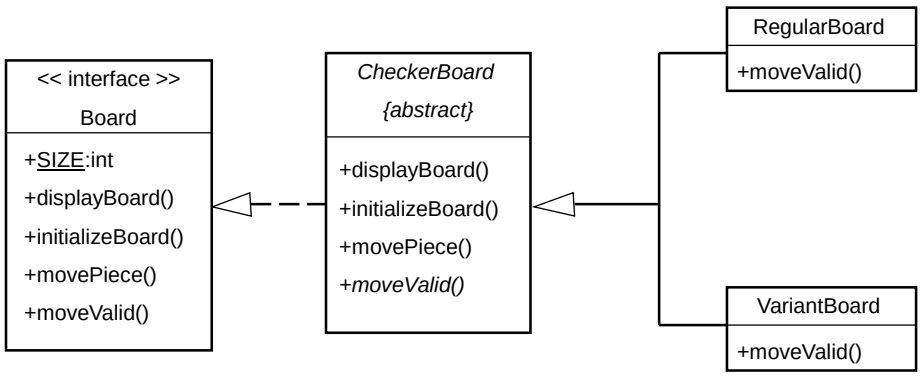
- Example[1]:

```
public abstract class BankAccount
{
   protected float balance;
   public void displayBalance()
   {
       System.out.println("Balance $" + balance);
   }
   public abstract void deductFees();
}
```

1) From "Big Java" by C. Horstmann pp. 449 – 500.

James Tam

Another Example For Using An Abstract Class



James Tam

You Should Now Know

- What are interfaces/types
  - How do types differ from classes
  - How to implement and use interfaces in Java
  - When interfaces should be employed
- What are abstract classes in Java and how do they differ from non-abstract classes and interfaces.
  - When to employ abstract classes vs. interfaces vs. 'regular' classes
- How to read/write UML notations for abstract classes and interfaces.
- What does multiple inheritance from multiple implementations

James Tam

Copyright Notification

- "Unless otherwise indicated, all images in this presentation are used with permission from Microsoft."

James Tam