COMP 401
MULTIPLE INHERITANCE: MOTIVATION AND ISSUES

Instructor: Prasun Dewan
Prerequisite

- Inheritance Abstract Classes
PROBLEM

- A modification to the course problem.
- Want to gather statistics on how many times a course was searched
  - How many times getTitle() was called.
public interface Course {
    public String getTitle();
    public String getDepartment();
    public int getNumber();
}
public interface LoggedCourse extends Course {
    public int getNumberOfQueries();
}

LOGGED COURSE INTERFACE
public abstract class ACourse {
    String title, dept;
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
    }
    public String getTitle() {
        return title;
    }
    public String getDepartment() {
        return dept;
    }
}
public abstract class ALoggedCourse extends ACourse implements LoggedCourse {
    int numberOfQueries = 0;
    public ALoggedCourse (String theTitle, String theDept) {
        super (theTitle, theDept);
    }
    public int getNumberOfQueries() {
        return numberOfQueries;
    }
    public String getTitle() {
        String retVal = super.getTitle();
        numberOfQueries++;
        return retVal;
    }
}
public interface FreshmanSeminar extends Course {
    public final int SEMINAR_NUMBER = 6;
}
Logged Freshman Seminar Interface?

public interface FreshmanSeminar extends Course {
    public final int SEMINAR_NUMBER = 6;
}

public interface LoggedCourse extends Course {
    public int getNumberOfQueries();
}

public interface LoggedFreshmanSeminar extends LoggedCourse {
    public final int SEMINAR_NUMBER = 6;
}

public interface LoggedFreshmanSeminar extends FreshmanSeminar {
    public int getNumberOfQueries();
}
**Single Inheritance Alternative 1**

```java
public interface Course {
    public String getTitle();
    public String getDepartment();
    public int getNumber();
}

public interface LoggedCourse extends Course {
    public int getNumberOfQueries();
}

public interface FreshmanSeminar extends Course {
    public final int SEMINAR_NUMBER = 6;
}

public interface LoggedFreshmanSeminar extends LoggedCourse {
    public final int SEMINAR_NUMBER = 6;
}
```
public interface Course {
    public String getTitle();
    public String getDepartment();
    public int getNumber();
}

public interface LoggedCourse extends Course {
    public int getNumberOfQueries();
}

public interface FreshmanSeminar extends Course {
    public final int SEMINAR_NUMBER = 6;
}

public interface LoggedFreshmanSeminar extends FreshmanSeminar {
    public int getNumberOfQueries();
}
**Multiple Inheritance**

```
public interface Course {
    public String getTitle();
    public String getDepartment();
    public int getNumber();
}

public interface LoggedCourse
    extends Course {
    public int getNumberOfQueries();
}

public interface FreshmanSeminar
    extends Course {
    public final int SEMINAR_NUMBER = 6;
}

public interface LoggedFreshmanSeminar
    extends FreshmanSeminar, LoggedCourse {
}
```
Multiple Class Inheritance?

```java
public class AFreshmanSeminar extends ACourse
    implements Course {
    public ALoggedFreshmanSeminar(String theTitle, String theDept) {
        super (theTitle, theDept);
    }
    public int getNumber() {
        return SEMINAR_NUMBER;
    }
}
```

```java
public class ALoggedFreshmanSeminar implements LoggedFreshmanSeminar {
    extends AFreshmanSeminar, ALoggedCourse
    {
    ...
    }
```
public class AFreshmanSeminar extends ACourse {
    implements Course {
        public ALoggedFreshmanSeminar(String theTitle, String theDept) {
            super (theTitle, theDept);
        }
        public int getNumber() {
            return SEMINAR_NUMBER;
        }
    }
}

public abstract class ALoggedCourse extends ACourse implements LoggedCourse {
    public int numberOfQueries = 0;
    public ALoggedCourse (String theTitle, String theDept) {
        super (theTitle, theDept);
    }
    public String getTitle() {
        String retVal = super.getTitle();
        numberOfQueries++;
        return retVal;
    }
    public int getNumberOfQueries() {
        return numberOfQueries;
    }
}

public class ALoggedFreshmanSeminar extends AFreshmanSeminar, ALoggedCourse implements LoggedFreshmanSeminar {
    public ALoggedFreshmanSeminar String theTitle, String theDept) {
        super (theTitle, theDept);
    }
}

numberOfQueries uninitialized
Calling Both Constructors

```java
public class AFreshmanSeminar extends ACourse
    implements Course {
    public ALoggedFreshmanSeminar(String theTitle, String theDept) {
        super (theTitle, theDept);
    }
    public int getNumber() {
        return SEMINAR_NUMBER;
    }
}

public abstract class ALoggedCourse extends ACourse
    implements LoggedCourse {
    int numberOfQueries = 0;
    public ALoggedCourse (String theTitle, String theDept) {
        super (theTitle, theDept);
    }
    public String getTitle() {
        String retVal = super.getTitle();
        numberOfQueries++;
        return retVal;
    }
    public int getNumberOfQueries() {
        return numberOfQueries;
    }
}

public class ALoggedFreshmanSeminar
    extends AFreshmanSeminar, ALoggedCourse implements LoggedFreshmanSeminar {
    public ALoggedFreshmanSeminar(String theTitle, String theDept) {
        super (theTitle, theDept);
    }
}
```

ACourse constructor called twice
public abstract class ACourse {
    String title, dept;
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
    }
    public String getTitle() {
        return title;
    }
    public String getDepartment() {
        return dept;
    }
}

ACourse constructor called twice

Does not cause a problem
public abstract class ACourse {
    String title, dept;
    static int numCourses;
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        numCourses++;
        System.out.println("A Course Created");
    }
    public String getTitle() {
        return title;
    }
    public String getDepartment() {
        return dept;
    }
    public static int getNumCourses() {
        return numCourses;
    }
}

constructor called twice
Does cause a problem
Constructor is not idempotent
**Idempotent vs. Non Idempotent**

```java
public ACourse (String theTitle, String theDept) {
    title = theTitle;
    dept = theDept;
}
```

```java
public ACourse (String theTitle, String theDept) {
    title = theTitle;
    dept = theDept;
    numCourses++;
    System.out.println("A Course Created");
}
```

**Idempotent operation:**
Calling the operation 1 time has the same effect as calling it successfully an arbitrary number of times.

**All functions without side effects are idempotent.**

**Some procedures are also idempotent.**
Which superclass constructors should be called?

Calling constructor of only one of the superclasses can leave variables of other superclasses uninitialized

Calling constructors of all superclasses can result in constructor of some common superclass of the superclasses to be called twice

If constructor is not idempotent, we are likely to get results the writer of the idempotent constructor did not expect

What if we forced constructors to be idempotent – can we allow multiple inheritance?
What if we forced constructors to be idempotent and call all inherited constructors – can we allow multiple inheritance?

Non constructor methods?
public class AFreshmanSeminar extends ACourse implements Course {
    public ALoggedFreshmanSeminar(String theTitle, String theDept) {
        super(theTitle, theDept);
    }
    public int getNumber() {
        return SEMINAR_NUMBER;
    }
    public void print() {
        System.out.println("Seminar");
    }
}

Call all inherited methods and force them also to be idempotent?

If we special case non constructors, which one of these should be called?

public class ALoggedCourse extends ACourse implements LoggedCourse {
    int numberOfQueries = 0;
    public ALoggedCourse(String theTitle, String theDept) {
        super(theTitle, theDept);
    }
    public String getTitle() {
        String retVal = super.getTitle();
        numberOfQueries++;
        return retVal;
    }
    public int getNumberOfQueries() {
        return numberOfQueries;
    }
    public void print() {
        System.out.println("Logged");
    }
}

public class ALoggedFreshmanSeminar extends AFreshmanSeminar, ALoggedCourse implements LoggedFreshmanSeminar {
    public ALoggedFreshmanSeminar(String theTitle, String theDept) {
        super(theTitle, theDept);
    }
}
Inheriting Non Constructor Functions

Function cannot return multiple values, so must have different rules for them and constructors

Which function should be inherited?

public abstract class ALoggedCourse extends ACourse

public class ALoggedFreshmanSeminar extends AFreshmanSeminar

public class ALoggedFreshmanSeminar extends AFreshmanSeminar, ALoggedCourse implements LoggedFreshmanSeminar

Function cannot return multiple values, so must have different rules for them and constructors
**Dominant-Recessive Rules: Choose First Superclass?**

```java
public class ALoggedFreshmanSeminar extends AFreshmanSeminar implements LoggedCourse {
    public ALoggedFreshmanSeminar(String theTitle, String theDept) {
        super (theTitle, theDept);
    }
    public int getNumber() {
        return SEMINAR_NUMBER;
    }
    public String toString() {
        return "Seminar";
    }
}
```

Confusing to use order as in imports

Java does not support multiple inheritance for interfaces but not classes

```java
public abstract class ALoggedCourse extends ACourse implements LoggedCourse {
    int numberOfQueries = 0;
    public ALoggedCourse(String theTitle, String theDept) {
        super (theTitle, theDept);
    }
    public String getTitle() {
        String retVal = super.getTitle();
        numberOfQueries++;
        return retVal;
    }
    public int getNumberOfQueries() {
        return numberOfQueries;
    }
    public String toString() {
        return "Logged";
    }
}
```

```java
public class ALoggedFreshmanSeminar extends AFreshmanSeminar, ALoggedCourse implements LoggedFreshmanSeminar {
    public ALoggedFreshmanSeminar(String theTitle, String theDept) {
        super (theTitle, theDept);
    }
}
```
INHERITING THROUGH DIFFERENT PATHS

```java
public interface Course {
    public String getTitle();
    public String getDepartment();
    public int getNumber();
}
```

```java
public interface LoggedCourse extends Course {
    public int getNumberOfQueries();
}
```

```java
public interface FreshmanSeminar extends Course {
    public final int SEMINAR_NUMBER = 6;
}
```

```java
public interface LoggedFreshmanSeminar extends FreshmanSeminar, LoggedCourse {
}
```

Same method headers inherited twice, no problem
public interface Course {
    public String getTitle();
    public String getDepartment();
    public int getNumber();
}

public interface LoggedCourse extends Course {
    public int getNumberOfQueries();
    public String getTitle();
}

public interface FreshmanSeminar extends Course {
    public final int SEMINAR_NUMBER = 6;
}

public interface LoggedFreshmanSeminar extends FreshmanSeminar, LoggedCourse {
}

Equal method headers inherited twice
Interface = set of headers, removes duplicates
Inheriting Headers with Different Return Types

```java
public interface Course {
    public String getTitle();
    public String getDepartment();
    public int getNumber();
    public void initTitle(String initVal);
}

public interface LoggedCourse extends Course {
    public int getNumberOfQueries();
    public void initTitle(String initVal);
}

public interface FreshmanSeminar extends Course {
    public final int SEMINAR_NUMBER = 6;
}

public interface LoggedFreshmanSeminar extends FreshmanSeminar, LoggedCourse {}
```

Overload conflict, inheritance not allowed
**Multiple inheritance rules**

- Allow interfaces to inherit multiple times
  - Only headers are inherited
- Do not allow classes to inherit multiple times (Java)
  - Can be confusing to programmers
- Other solutions
  - Allow multiple inheritance if ambiguity does not arise
  - If ambiguity arises indicate which implementation is used (C++)
    - ALoggedCourse.toString() vs AFreshmanSeminar.toString()
  - Choose one or none of the bodies if problem arises
    - Based on order in extends clause?
    - extends ALoggedCourse, AFreshmanSeminar
public class ALoggedFreshmanSeminar extends ALoggedCourse implements LoggedFreshmanSeminar {
    public ALoggedFreshmanSeminar(String theTitle, String theDept) {
        super (theTitle, theDept);
        
        int numberOfQueries = 0;
        
        public int getNumber() {
            return SEMINAR_NUMBER;
        }
        
        public String getTitle() {
            String retval = super.getTitle();
            numberOfQueries++;
            return retval;
        }
        
        public int getNumberOfQueries() {
            return numberOfQueries;
        }
    }
}

public class AFreshmanSeminar extends ACourse implements Course {
    public AFreshmanSeminar(String theTitle, String theDept) {
        super (theTitle, theDept);
    }
    
    public int getNumber() {
        return SEMINAR_NUMBER;
    }
}

public class ALoggedFreshmanSeminar extends ALoggedCourse implements LoggedFreshmanSeminar {
    public ALoggedFreshmanSeminar(String theTitle, String theDept) {
        super (theTitle, theDept);
    }
    
    public int getNumber() {
        return SEMINAR_NUMBER;
    }
}
Reduced Polymorphism?

static void processClass (AFreshmanSeminar course) {
    ...
}

processClass (new ALoggedFreshmanSeminar("Random Thoughts", "CS"));

static void processInterface (FreshmanSeminar course) {
    ...
}

processInterface (new ALoggedFreshmanSeminar("Random Thoughts", "CS"));


**Single inheritance => reduced polymorphism?**

- Yes, if classes used to type variables
- Interfaces offer solution to lack of multiple (class) inheritance in Java
- Most programmers do not realise other uses.
- In text books, interfaces introduced and used only in problems requiring multiple inheritance
Implementing Multiple Interfaces vs. Single Extended Interface

```java
public class ALoggedFreshmanSeminar extends ALoggedCourse
    implements LoggedFreshmanSeminar {
    ...
}
```

```java
processInterface (new ALoggedFreshmanSeminar("Random Thoughts", "CS"));
```

```java
public class ALoggedFreshmanSeminar extends ALoggedCourse
    implements LoggedCourse, FreshmanSeminar {
    ...
}
```

```java
processInterface ((FreshmanSeminar) new ALoggedFreshmanSeminar("Random Thoughts", "CS"));
```

```java
static void processInterface (FreshmanSeminar course) {
    ...
}
```
IMPLEMENTING MULTIPLE INTERFACES VS. SINGLE INTERFACE

- Does not require casting to use different interfaces of same object
- Modulo overloading problems
  - These arise whenever an object can be typed in multiple ways.
  - A compile time issue – these casts are needed only at compile time and do not lead to runtime errors.
HOW TO AVOID CODE DUPLICATION

public class AFreshmanSeminar extends ACourse
    implements Course {
    public ALoggedFreshmanSeminar(String theTitle, String theDept) {
        super (theTitle, theDept);
    }
    public int getNumber() {
        return SEMINAR_NUMBER;
    }
}

public class ALoggedCourse extends ACourse
    implements LoggedCourse {
    int numberOfQueries = 0;
    public ALoggedCourse (String theTitle, String theDept) {
        super (theTitle, theDept);
    }
    public String getTitle() {
        String retVal = super.getTitle();
        numberOfQueries++;
        return retVal;
    }
    public int getNumberOfQueries() {
        return numberOfQueries;
    }
}

public class ALoggedFreshmanSeminar
    extends AFreshmanSeminar, ALoggedCourse
    implements LoggedFreshmanSeminar {
    public ALoggedFreshmanSeminar String theTitle, String theDept) {
        super (theTitle, theDept);
    }
    public int getNumber() {
        return SEMINAR_NUMBER;
    }
}
**How to Avoid Code Duplication?**

```java
public class AFreshmanSeminar extends ACourse implements Course {
    public ALoggedFreshmanSeminar(String theTitle, String theDept) {
        super(theTitle, theDept);
    }
    public int getNumber() {
        return SEMINAR_NUMBER;
    }
}
```

```java
public abstract class ALoggedCourse extends ACourse implements LoggedCourse {
    int numberOfQueries = 0;
    public ALoggedCourse (String theTitle, String theDept) {
        super (theTitle, theDept);
        numberOfQueries = super.getTitle();
        numberOfQueries++;
        return numberOfQueries;
    }
    public String getTitle() {
        String retVal = super.getTitle();
        numberOfQueries += super.getTitle();
        return retVal;
    }
    public int getNumberOfQueries() {
        return numberOfQueries;
    }
}
```

```java
public class ALoggedFreshmanSeminar extends ALoggedCourse implements LoggedFreshmanSeminar {
    FreshmanSeminar seminar;
    public ALoggedFreshmanSeminar (String theTitle, String theDept) {
        super (theTitle, theDept);
        seminar = new AFreshmanSeminar (theTitle, theDept);
    }
    public int getNumber() {
        return seminar.getNumber();
    }
}
```