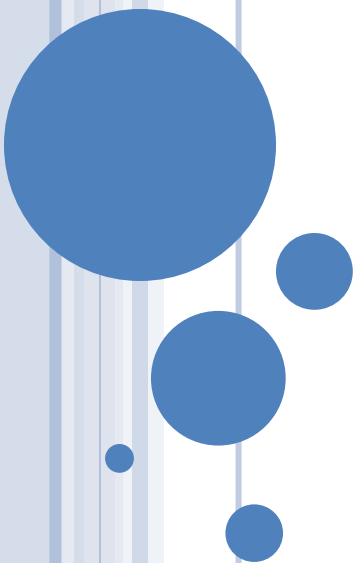




COMP 401

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.

COURSE INTERFACE

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



LOGGED COURSE INTERFACE

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

ACOURSE

```
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;  
    }  
}
```

ALOGGEDCOURSE

```
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;
    }
}
```

FRESHMAN SEMINAR INTERFACE

```
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();  
}
```

Code Duplication

SINGLE INHERITANCE ALTERNATIVE 1

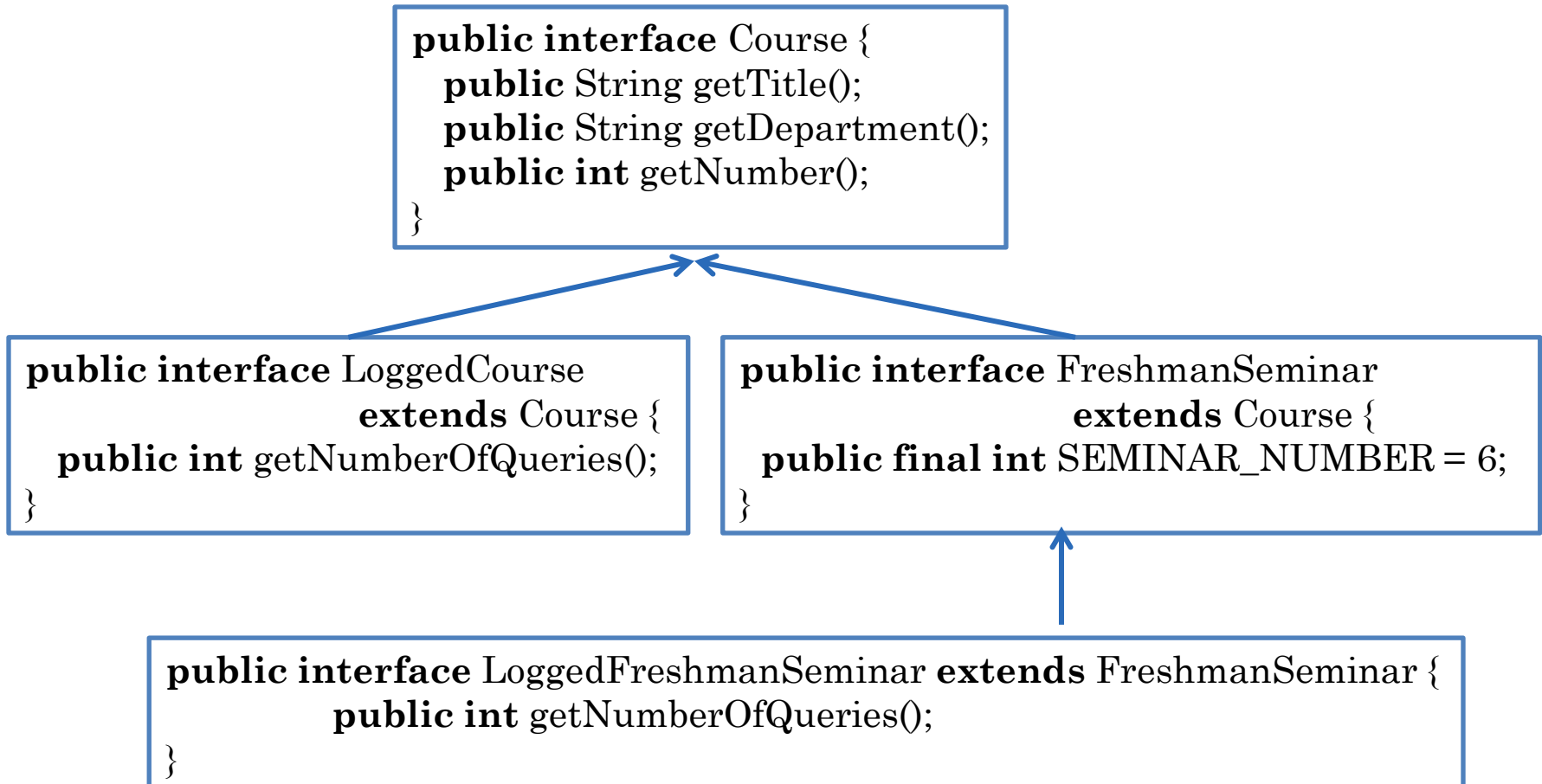
```
public interface Course {  
    public String getTitle();  
    public String getDepartment();  
    public int getNumber();  
}
```

```
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;  
}
```

SINGLE INHERITANCE ALTERNATIVE 2



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  
{  
}
```

Empty
interface

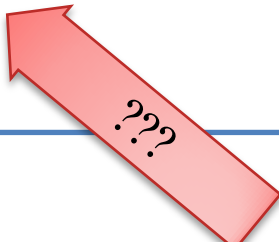
Multiple
Inheritance

MULTIPLE CLASS INHERITANCE?

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



```
public class ALoggedFreshmanSeminar
    extends AFreshmanSeminar, ALoggedCourse implements LoggedFreshmanSeminar
{
    ...
}
```



CALLING AFRESHMANSEMINAR CONSTRUCTOR

```
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;
    }
}
```

numberOfQueries
uninitialized

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

CALLING BOTH CONSTRUCTORS

```
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;
    }
}
```

ACourse constructor
called twice

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

ACOURSE

```
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

ALTERNATIVE ACourse

```
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

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

```
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

MULTIPLE CLASS INHERITANCE ISSUES

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?

MULTIPLE CLASS INHERITANCE ISSUES

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

Non constructor methods?

INHERITING NON CONSTRUCTOR FUNCTIONS

```
public class AFreshmanSeminar extends ACourse {
    public String getTitle() {
        return "Seminar";
    }
}

public abstract class ALoggedCourse extends ACourse {
    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);
    }
}
```

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

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

INHERITING NON CONSTRUCTOR FUNCTIONS

```
public class AFreshmanSeminar extends ACourse
```

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

```
public  
super  
{  
}
```

```
public  
return
```

Which function should be inherited?

```
public String toString() {  
    return "Seminar";  
}
```

```
public abstract class ALoggedCourse extends ACourse
```

```
super (theTitle, theDept),  
{  
}
```

```
public String getTitle() {  
    String retVal = super.getTitle();  
    numberOfQueries++;  
    return retVal;  
}
```

```
public int getNumberOfQueries() {  
    return numberOfQueries;  
}
```

```
public String toString() {  
    return "Logged";  
}
```

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

DOMINANT-RECESSIVE RULES: CHOOSE FIRST SUPERCLASS?

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

Confusing to use order as in imports

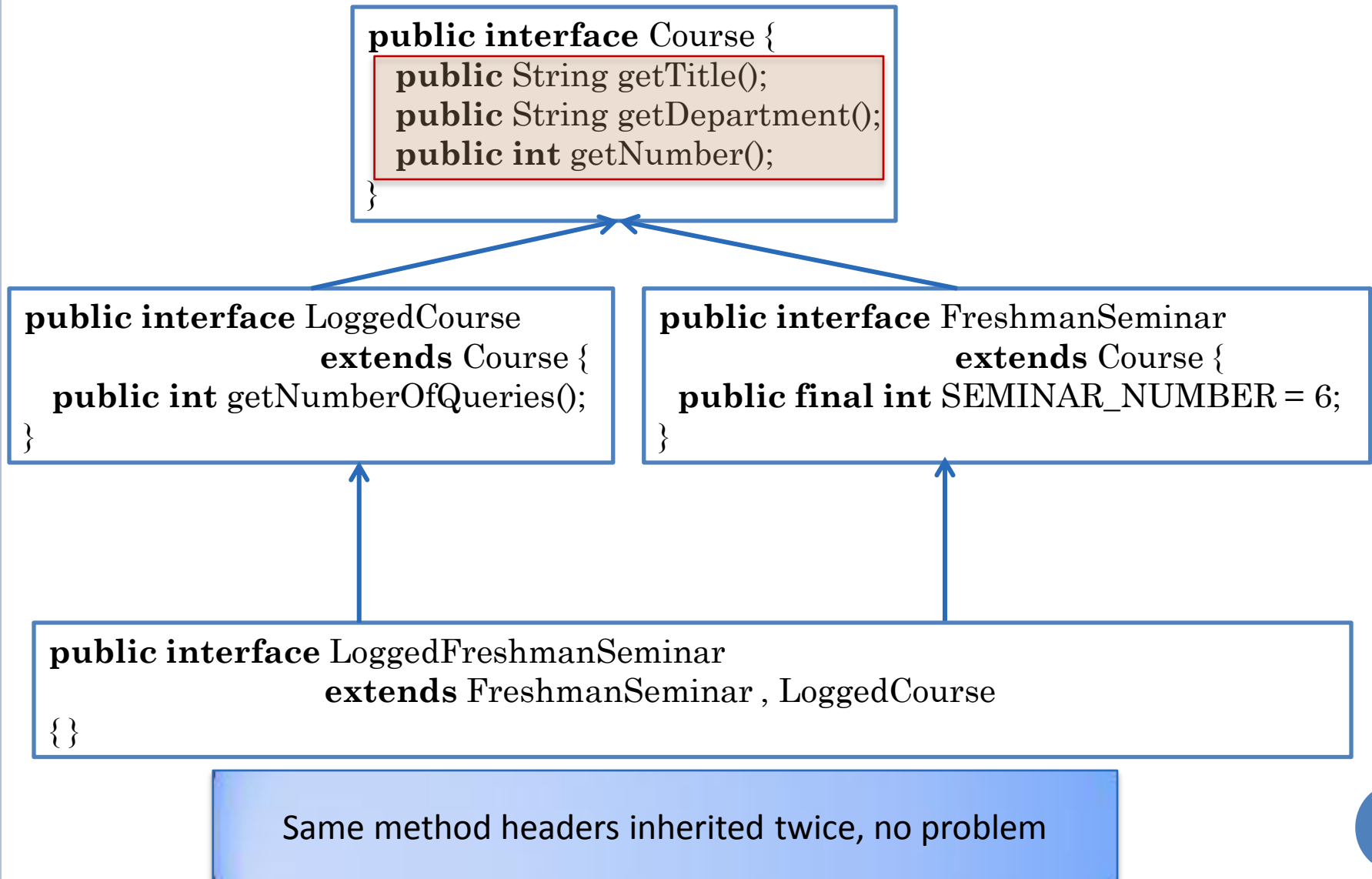
```
public abstract class ALoggedCourse extends ACourse
```

Java does not support multiple inheritance
for interfaces but not classes

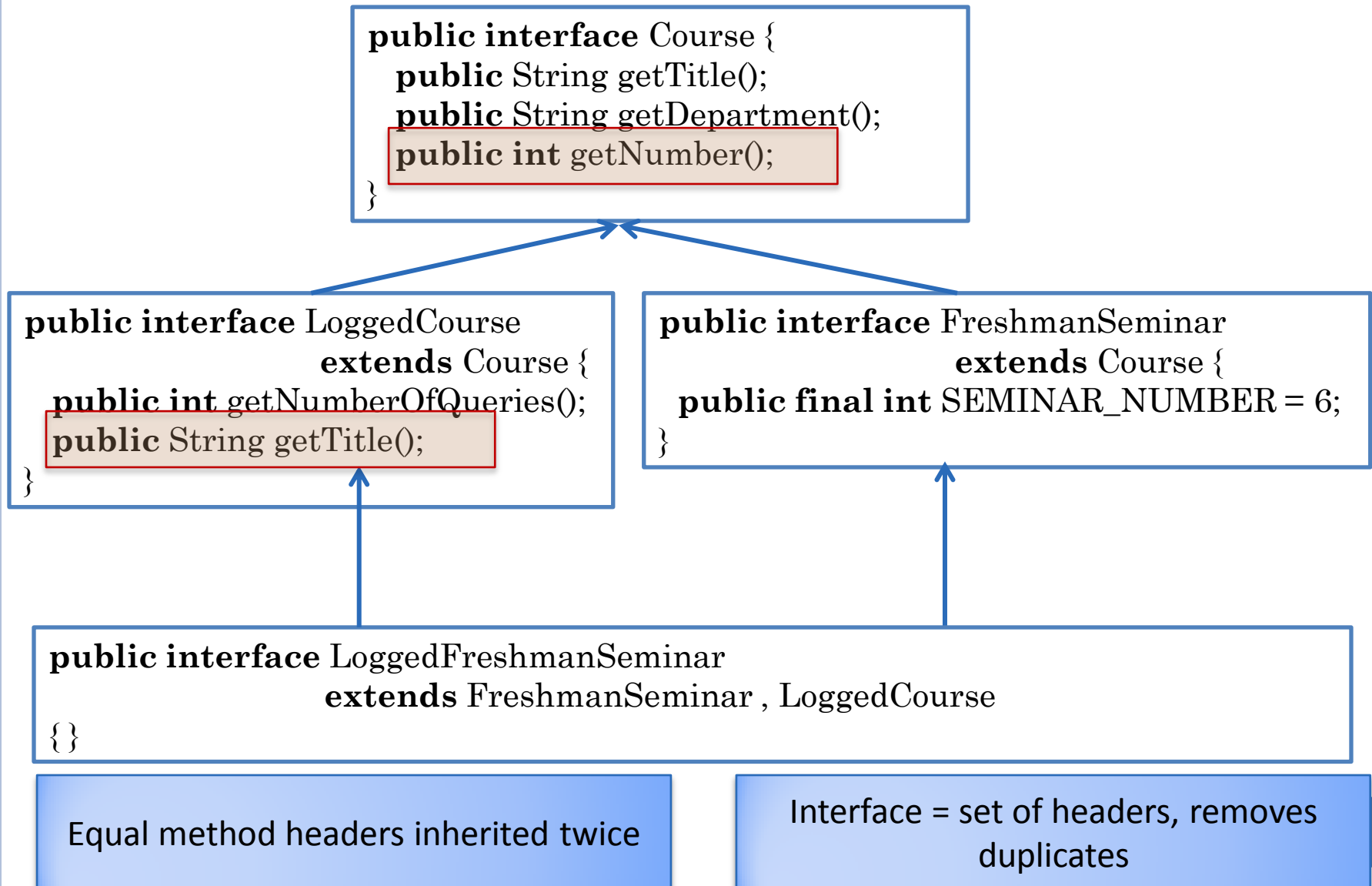
```
    public String getTitle() {
        String retVal = super.getTitle();
        numberOfQueries++;
        return retVal;
    }
    public int getNumberOfQueries() {
        return numberOfQueries;
    }
    public String toString() {
        return "Logged";
    }
}
```

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

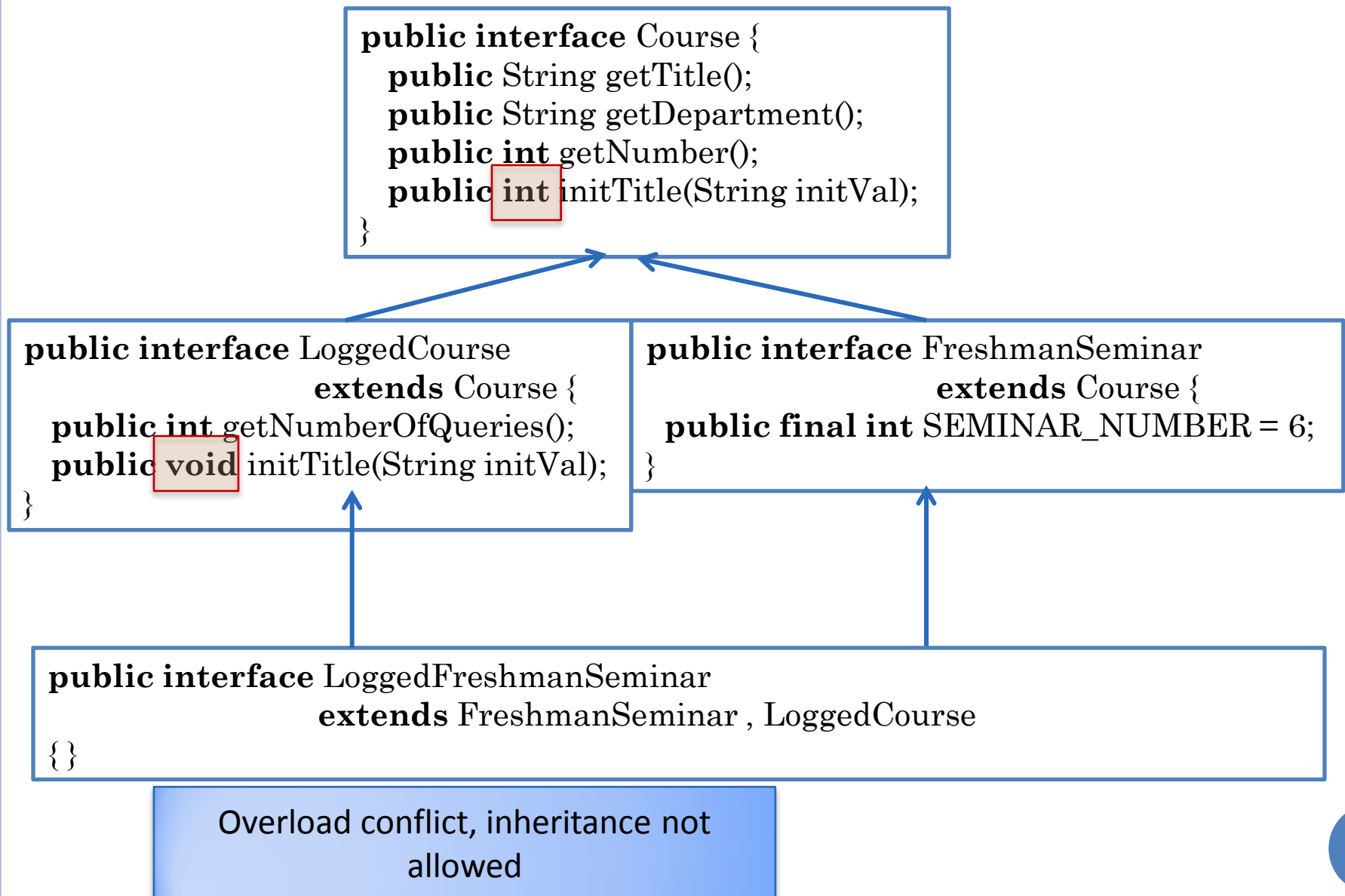
INHERITING THROUGH DIFFERENT PATHS



INHERITING EQUAL HEADERS



INHERITING HEADERS WITH DIFFERENT RETURN TYPES



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`

REPEATING CODE

```
public class AFreshmanSeminar extends ACourse  
    implements Course {
```

```
    public ALoggedFreshmanSeminar(String theTitle, String theDept) {  
        super (theTitle, theDept);  
    }  
    public int getNumber() {  
        return SEMINAR_NUMBER;  
    }  
}
```

```
    public ACourse
```

```
    get theDept) {
```

```
        String retVal = super.getTitle(),  
        numberOfQueries++;  
        return retVal;  
    }  
    public int getNumberOfQueries() {  
        return numberOfQueries;  
    }  
}
```



```
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

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

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

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

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

```
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 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;  
    }  
}
```

IS-A

IS-A

```
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?

