PREREQUISITE

- Inheritance Abstract Classes

.
MORE INHERITANCE
public class ARegularCourse extends ACourse implements Course {
    int courseNum;
    public ARegularCourse (String theTitle, String theDept, int theCourseNum) {
        super (theTitle, theDept);
        courseNum = theCourseNum;
    }
    public int getNumber() {
        return courseNum;
    }
}
public class ARegularCourse extends ACourse implements Course {
    int courseNum;
    public ARegularCourse (String theTitle, String theDept, int theCourseNum) {
        courseNum = theCourseNum;
        super (theTitle, theDept);
    }
    public int getNumber() {
        return courseNum;
    }
}

Super call must be first statement in constructor but not other methods.

Subclass may want to override initialization in super class

Superclass vars initialized before subclass vars, which can use the former

Subclass vars not visible in superclass
public class ARegularCourse extends ACourse implements Course {
    int courseNum;
    public ARegularCourse (String theTitle, String theDept, int theCourseNum) {
        courseNum = theCourseNum;  
    }
    public int getNumber() {
        return courseNum;
    }
}

No super call!
public abstract class ACourse extends Object
{
    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 ACourse extends Object {
    String title, dept;
    public ACourse (String theTitle, String theDept) {
        super();
        title = theTitle;
        dept = theDept;
    }
    public String getTitle() {
        return title;
    }
    public String getDepartment() {
        return dept;
    }
}

Automatically inserted
public class ARegularCourse extends ACourse implements Course {
    int courseNum;
    public ARegularCourse (String theTitle, String theDept, int theCourseNum) {
        courseNum = theCourseNum;  // No super call!
    }
    public int getNumber() {
        return courseNum;
    }
}
public class ARegularCourse extends ACourse implements Course {
    int courseNum;
    public ARegularCourse (String theTitle, String theDept, int theCourseNum) {
        super();
        courseNum = theCourseNum;
    }
    public int getNumber() {
        return courseNum;
    }
}
ADVANCED INITIALIZATION
**Abstract Methods in Constructors**

```java
public abstract class ACourse implements Course {
    String title, dept;
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        // "innocuous" debugging statement added to ACourse
        System.out.println("New course created: "+"Title:"+ title + "Dept:"+ dept + "Number:" + getNumber());
    }

    public String getTitle() {
        return title;
    }

    public String getDepartment() {
        return dept;
    }

    abstract public int getNumber();
}
```
Abstract methods in constructors

getNumber() called before subclass constructor in ARegularCourse has initialized variables.
ABSTRACT METHODS IN CONSTRUCTORS

- Beware of abstract methods being called in constructors.
- They may access uninitialized variables in subclasses!
public class ARegularCourse extends ACourse {
    int courseNum = 99;
    public ARegularCourse (String theTitle, String theDept, int theCourseNum) {
        super (theTitle, theDept);
        courseNum = theCourseNum;
    }
    public int getNumber() {
        return courseNum;
    }
}
INITIALIZING DECLARATIONS

Try stepping into a constructor in a debugger.

Why correct value?

Initializations in declarations processed after superclass constructor returns
**CONSTANT INITIALIZATIONS**

- Done when constants allocated memory.
- (Should be) shared by all instances.
- When class is loaded in memory.
- Not when an instance created.
The variables declared in ARegularCourse and Course are allocated space but not initialized.
The constructor of ARegularCourse is called.
It calls the constructor of ACourse in its first statement.
The initializations in the declarations of the variables of ACourse are processed. In this case there are no initializations.

The constructor of ACourse is started.

The two instance variables of ACourse are assigned parameter values, “Intro. Prog.” And “COMP”.

The values of these variables are printed.

The method getNumber() of ARegularCourse is called and the default value of courseNum is printed.
The constructor of ACourse returns.
The initializations in the declarations of the variables of ARegularCourse are processed. In this case, courseNum is assigned the value 0.
Execution resumes in the constructor of ARegularCourse. The courseNum variable is assigned the parameter value, 14.
The constructor of ARegularCourse returns.
The new statement completes, returning the new instance to its caller.
**MULTIPLE CONSTRUCTORS**

```java
public abstract class ACourse implements Course {
    String title = "COMP";
    String dept = "Topics in Computer Science";

    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: Title:" + title + " Dept:" + dept + " Number: " + getNumber());
    }

    public ACourse () {
        System.out.println("New course created: Title:" + title + " Dept:" + dept + " Number: " + getNumber());
    }

    public String getTitle() {return title;}
    public String getDepartment() {return dept;}

    abstract public int getNumber();
}
```

**How to remove code duplication?**
public abstract class ACourse implements Course {
    String title = "COMP";
    String dept = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:"+ dept + " Number:" + getNumber());
    }
    public ACourse () {
        this (title, dept);
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}

This implies constructor of same class
```java
public abstract class ACourse implements Course {

    String title = "COMP";
    String dept = "Topics in Computer Science";

    public ACourse (String theTitle, String theDept) {
        super();
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:"+ dept + " Number:" + getNumber());
    }

    public ACourse () {
        this (title, dept);
    }

    public String getTitle() {return title;}
    public String getDepartment() {return dept;}

    abstract public int getNumber();
}
```

Super may have side effects, so cannot call super() before this()
Java complains that instance variables not initialized
CONSTRUCTOR CALLING CONSTRUCTOR

```java
public abstract class ACourse implements Course {
    String title = "COMP";
    String dept = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        this();
    }
    public ACourse () {
        System.out.println("New course created: " + "Title:" + title
        + " Dept:"+ dept + " Number: " + getNumber());
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}
```

Java complains that this() must be first call, as it results in super() being called
**CONSTRUCTOR CALLING CONSTRUCTOR**

```java
public abstract class ACourse implements Course {
    String title, dept;
    final String DEFAULT_DEPT = "COMP";
    final String DEFAULT_TITLE = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:" + dept + " Number: " + getNumber());
    }
    public ACourse () {
        this ("COMP", "Topics in Computer Science");
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}
```

No complaints.
CONSTRUCTOR CALLING CONSTRUCTOR

```
public abstract class ACourse implements Course {
    String title, dept;
    final String DEFAULT_DEPT = "COMP";
    final String DEFAULT_TITLE = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:" + dept + " Number: " + getNumber());
    }
    public ACourse () {
        this (DEFAULT_DEPT, DEFAULT_TITLE);
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}
```

Java complains that instance vars cannot be accessed.
POSSIBLE CONSTRUCTOR CALL SEMANTICS

- Rule so far: each constructor calls superclass constructor.
- Problem: then superclass initialized multiple times.
  - Not a problem if each initialization yields the same result.
  - Can have multiple constructors doing different initializations.
  - Same constructor may do different things on different invocations or may have side effects.
ACTUAL SEMANTICS

- A constructor may call another constructor as first statement.
- Super not called in that case.
- Only last call in constructor chain calls super.
- Hence no danger of super being initialized multiple times.
- But danger that constructor parameters may be uninitialized.
- Literal parameters ok.
- Constructor call must be first statement.
- Otherwise code executed before superclass initialized.
SOLUTION: INIT METHOD

```java
public abstract class ACourse implements Course {
    String title, dept;
    final String DEFAULT_DEPT = "COMP";
    final String DEFAULT_TITLE = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        init (theTitle, theDept)
    }
    public ACourse () {
        init (DEFAULT_DEPT, DEFAULT_TITLE);
    }
    void init (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:"+ dept + " Number: " + getNumber());
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}
```
ALLOWING INIT AFTER CONSTRUCTION

```java
public abstract class ACourse implements Course {
    String title = "Topics in Computer Science";
    String dept = "COMP";
    public ACourse (String theTitle, String theDept) {
        init (theTitle, theDept)
    }
    public ACourse () {
        init (DEFAULT_DEPT, DEFAULT_TITLE);
    }
    public void init (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:"+ dept + " Number: " + getNumber());
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}
```
public class ARegularCourse extends ACourse {
    int courseNum;
    public ARegularCourse (String theTitle, String theDept, int theCourseNum) {
        super (theTitle, theDept);
        courseNum = theCourseNum;
    }
    public ARegularCourse () {
    }
    public int getNumber() {
        return courseNum;
    }
}

Course course = new ARegularCourse();
course.init(“Meaning of Life”, “PHIL”);
public class ARegularCourse extends ACourse {
    int courseNum;
    public ARegularCourse (String theTitle, String theDept, int theCourseNum) {
        super (theTitle, theDept);
        courseNum = theCourseNum;
    }
    public ARegularCourse () {
    }
    public int getNumber() {
        return courseNum;
    }
    public void init (String theTitle, String theDept, int theCourseNum) {
        courseNum = theCourseNum;
    }
}

Course course =new ARegularCourse();
Course.init(“Meaning of Life”, “PHIL”, 999);
public class ARegularCourse extends ACourse {
    int courseNum;
    public ARegularCourse (String theTitle, String theDept, int theCourseNum) {
        super (theTitle, theDept);
        courseNum = theCourseNum;
    }
    public ARegularCourse () {
    }
    public int getNumber() {
        return courseNum;
    }
    public void init (String theTitle, String theDept, int theCourseNum) {
        init(theTitle, theCourseNum);
        courseNum = theCourseNum;
    }
}

Course course = new ARegularCourse();
Course.init(“Meaning of Life”, “PHIL”, 999);
TWO WAYS TO CONSTRUCT

- Initialized construction
  - `new ARegularCourse ("Intro. Prog", "COMP", 14);`
- Construction and then initialization
  - `(new ARegularCourse()).init("Intro Prog.", "COMP", 14)`
PARAMETERIZED CONSTRUCTORS

- Can create multiple parameterized constructors
  - To initializes some subset of instance variables.
  - Use default values assigned by initializing declarations for the rest.
    ```java
    public ACourse (String theTitle) {
        title = theTitle;
    }
    ```

- Add an init method for each parameterized constructor
INIT METHODS

- Allows initialization after object is created.
- Initializer can be different from creator
  - Abstract class may initialize
  - Concrete factory method may instantiate.
- Init methods can be in interfaces
- Init method(s) recommended but not required
public class AFreshmanSeminar extends ACourse {
    public AFreshmanSeminar (String theTitle, String theDept) {
        super (theTitle, theDept);
        title = theTitle;
    }
    public int getNumber() {
        return SEMINAR_NUMBER;
    }
}
abstract public class ACourse {
    String title, dept;
    public ACourse (String theTitle, String theDept) {
        //title = theTitle;
        dept = theDept;
        // “innocuous” debugging statement added to ACourse
        System.out.println("New course created: " + "Title:" + title + " Dept:" + dept + " Number: " + getNumber()}
    }
    public String getTitle() {
        return title;
    }
    public String getDepartment() {
        return dept;
    }
    abstract public int getNumber();
}
DISPLAY ABSTRACT COURSE

```
buzzard(136)%%!77
java main.ACourseDisplayer
New course created: Title:null Dept:COMP Number: 0
New course created: Title:null Dept:COMP Number: 0
New course created: Title:null Dept:COMP Number: 6
New course created: Title:null Dept:COMP Number: 6
Please enter course title:
Intro. Prog.
TITLE NUMBER
Intro. Prog. COMP14
Please enter course title:
```
public abstract class ACourse {
    String title, dept;
    public ACourse (String theTitle, String theDept) {
        //title = theTitle;
        dept = theDept;
        // “innocuous” debugging statement added to ACourse
        System.out.println("New course created: " + "Title:" + title + " Dept:" + dept + " Number: " + getNumber()}
    public String getTitle() {
        return title;
    }
    public String getDepartment() {
        return dept;
    }
}
public class ARegularCourse extends ACourse {
    int courseNum;
    String title;
    public ARegularCourse (String theTitle, String theDept, int theCourseNum) {
        super (theTitle, theDept);
        courseNum = theCourseNum;
        title = theTitle;
    }
    public int getNumber() {
        return courseNum;
    }
}
public class AFreshmanSeminar extends ACourse {

    String title;

    public AFreshmanSeminar (String theTitle, String theDept) {
        super (theTitle, theDept);
        title = theTitle
    }

    public int getNumber() {
        return SEMINAR_NUMBER;
    }

}
REDECLARING SUPERCLASS VARIABLES

```
buzzard(146)\% ^!77
java main.ACourseDisplayer
New course created: Title:null Dept:COMP Number: 0
New course created: Title:null Dept:COMP Number: 0
New course created: Title:null Dept:COMP Number: 6
New course created: Title:null Dept:COMP Number: 6
Please enter course title:
Intro. Prog.
Exception in thread "main" java.lang.NullPointerException
  at collections.ACourseList.matchTitle(ACourseList.java:29)
  at main.ACourseDisplayer.main(ACourseDisplayer.java:22)
buzzard(147)\%
```
public abstract class ACourse implements Course {
    String title, dept;
    public ACourse (String theTitle, String theDept) {
        //title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:"+ dept + " Number: " + getNumber());
    }
    public String getTitle() {
        return title;
    }
    public String getDepartment() {
        return dept;
    }
    abstract public int getNumber();
}
RE-DECLARING SUBCLASS VARIABLES

- Happens accidentally when class is re-factored manually to move subclass variables to super-classes.
- Original variable in subclass remains.
- Beware!
- Use Eclipse refactor→move command when possible.
EXTRA SLIDES
static void print (Course course) {
    if (course instanceof ARegularCourse)
        printHeader ((ARegularCourse) course);
    else if (course instanceof AFreshmanSeminar)
        printHeader ((AFreshmanSeminar) course);
    System.out.println(
        course.getTitle() + " " +
        course.getDepartment() +
        course.getNumber() );
}
static void printHeader (ARegularCourse course) {
    System.out.print(“Regular Course: ”);
}
static void printHeader (AFreshmanSeminar course) {
    System.out.print(“Freshman Seminar: ”);
}
public abstract class ACourse implements Course {
    String title = "Topics in Computer Science";
    String dept = "COMP";
    public ACourse (String theTitle, String theDept) {
        init (theTitle, theDept);
        init();
    }
    public ACourse () { init();}
    public void init (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
    }
    void init() {
        System.out.println("New course created: " + "Title:" + title + " Dept:"+ dept + " Number: " + getNumber());
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}
SOME SUPERTYPES IN EXAMPLES
Abstract Classes in java.util?
Abstract Classes in Java.util

AbstractCollection

AbstractSet

AbstractList

AbstractSequentialList

Collection

Set

List

HashSet

Vector

ArrayList

LinkedList

implements

deextends
POLYMORPHISM VS. OVERLOADING AND DYNAMIC DISPATCH

Create polymorphic code when you can
- In overloading and dynamic dispatch, multiple implementations associated with each method name.
- In polymorphic case, single implementation.
  - Use interfaces rather than classes as types of parameters
  - Use supertypes rather than subtypes as types of parameters

Polymorphism vs. Overloading
- Polymorphism: single print (Course course)
- Overloading: print (ARegularCourse course) and print (AFreshmanSeminar course) with same implementation.

Polymorphism vs. Dynamic Dispatch
- Polymorphism: single getTitle() in ACourse
- Dynamic dispatch: getTitle() in AFreshmanSeminar() and getTitle() in ARegularCourse() with same implementation.
POLYMORPHISM VS. OVERLOADING AND DYNAMIC DISPATCH

- Cannot always create polymorphic method.
  - `getNumber()` for `ARegularCourse` and `AFreshmanSeminar` do different things.
  - `print(Course course)` and `print (CourseList courseList)` do different things.
- When polymorphism not possible try overloading and dynamic dispatch.
OVERLOADING VS. DYNAMIC DISPATCH

- Overloading:
  - Object is parameter

- Dynamic dispatch:
  - Object is target

- Method in object class vs. external class
  - Program decomposition issue
  - `print(Course), print (CourseList)` definitions overloaded
  - `AFreshmanSeminar.getCourseNumber(), ARegularCourse.getCourseNumber()` dynamically dispatched

- Overload resolution looks at multiple parameter types
  - More general and used for that reason also

- Dynamic dispatch is done at runtime
  - Can be used for that reason
<table>
<thead>
<tr>
<th>size</th>
<th>array</th>
<th>size</th>
<th>array</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>James Dean</td>
<td>2</td>
<td>James Dean</td>
</tr>
<tr>
<td></td>
<td>Joe Doe</td>
<td></td>
<td>Jane Smith</td>
</tr>
<tr>
<td></td>
<td>Jane Smith</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Physical and Computer Inheritance

- Physical Object
  - Animal
    - Mammal
      - Primate
        - Human
  - Rock
- Implicit Extension
  - AString History
  - AString Database
- Object
  - String
public void addElement(Object element) {
    if (isFull())
        System.out.println("Adding item to a full history");
    else {
        contents[size] = element;
        size++;
    }
}

public void addElement(String element) {
    if (member(element)) return;
    super.addElement(element);
}
Point p1 = new ACartesianPoint(200, 200);

ACartesianPoint@8 8

Point p1 16

Point p2 48

p2 == p2 → true

ACartesianPoint@8

P1 P2

But not necessary.
abstract class ACourse {
    String title, dept;
    public ACourse(String theTitle, String theDept) {
        super();
        title = theTitle;
        dept = theDept;
    }
    public String getTitle() {
        return title;
    }
    public String getDepartment() {
        return dept;
    }
}
public abstract class ACourse implements Course {
    String title, dept;
    public ACourse (String theTitle, String theDept) {
        super();
        title = theTitle;
        dept = theDept;
    }
    public String getTitle() {
        return title;
    }
    public String getDepartment() {
        return dept;
    }
    abstract public int getNumber();
}
ABSTRACT METHOD

- Declared only in abstract classes
- **Keyword:** abstract
- No body
- Each (direct or indirect) subclass must implement abstract methods defined by an abstract class.
- Much like each class must implement the methods defined by its interface(s).
COURSE DISPLAYER USER INTERFACE

Please enter course title:
Intro. Prog.
TITLE NUMBER
Intro. Prog. COMP14
Please enter course title:
Comp. Animation
TITLE NUMBER
Comp. Animation COMP6
Please enter course title:
Lego Robots
TITLE NUMBER
Lego Robots COMP6
Please enter course title:
Meaning of Life
Sorry, this course is not offered.
Please enter course title:
Found. of Prog.
TITLE NUMBER
Found. of Prog. COMP114
Please enter course title:
Main class: Filling List

static CourseList courses = new ACourseList();
static void fillCourses() {
    courses.addElement(new ARegularCourse("Intro. Prog.", "COMP", 14));
    courses.addElement(new ARegularCourse("Found. of Prog.", "COMP", 114));
    courses.addElement(new AFreshmanSeminar("Comp. Animation", "COMP"));
    courses.addElement(new AFreshmanSeminar("Lego Robots", "COMP"));
}
JAVA VECTORS, ARRAY LISTS AND ITERATORS

- Just like collections we defined
- Except they can store and iterate arbitrary objects
Vectors (Java.util.Vector)

Vector v;

v = new Vector();
v.addElement("James Dean")
v.addElement("Joe Doe")
v.addElement(new ACartesianPoint(5, 5))
v.addElement(new Vector())
v.addElement(5)
**Important Methods of Class Vector**

- `public final int size()`
- `public final Object elementAt(int index)`
- `public final void addElement(Object obj)`
- `public final void setElementAt(Object obj, int index)`
- `public final void insertElementAt(Object obj, int index)`
- `public final boolean removeElement(Object obj)`
- `public final void removeElementAt(int index)`
- `public final int indexOf(Object obj)`
- `public final Enumeration elements()`
Methods of Interface Enumeration (java.util.Enumeration)

- `public boolean hasMoreElements();`
- `public Object nextElement();`

```java
Enumeration elements = vector.elements();
while (elements.hasMoreElements())
    System.out.println(elements.nextElement());
```
METHODS OF INTERFACE NUMERATION (JAVA.UTIL.ENUMERATION)

- public boolean hasMoreElements();
- public Object nextElement();

```java
for (Enumeration elements = vector.elements();
    elements.hasMoreElements();)
    System.out.println(elements.nextElement());
```
String history user

Vector v = new Vector();

Vector v = new Vector();
v.addElement("Joe Doe");
v.addElement("John Smith");
v.addElement(new Vector());
v.removeElementAt(0);
String history user
StringHistory stringHistory = new AStringHistory()

AStringHistory instance
Vector v = new Vector();

Vector Instance
size(): \(\rightarrow\) int

elementAt():
index \(\rightarrow\) String

addElement():
String \(\rightarrow\) void

removeElementAt():
Object \(\rightarrow\) void

stringHistory.addElement("Joe Doe");
stringHistory.addElement\((\text{new Vector()})\)
import java.util.Vector;

public class AStringHistory implements StringHistory {

    Vector contents = new Vector();

    public void addElement (String s) {
        contents.addElement(s);
    }

    public String elementAt (int index) {
        return (String) contents.elementAt(index);
    }

    public int size() {
        return contents.size();
    }
}

Simply converts types
**Adapter Pattern**

- Degree of adaptation undefined.
- Methods offered to client
  - Adapted name
  - Adapted type
**Adapter Pattern**

- Degree of adaptation undefined.
- Methods offered to client
  - Adapted name
  - Adapted type
**Parser Structure**

- Each production associated with a parser method.
- Parser method returns object associated with LHS of production.
- Usually at start of method execution
  - Prefix of unconsumed input should be legal phrase derived from LHS
  - Unless calling method consumed one or more tokens of the phrase to choose an alternative rule
- Such a parser called: recursive descent parser
- Illustrates top-down programming
public class ARegularCourse extends ACourse {
    int courseNum;
    public ARegularCourse (String theTitle, String theDept, int theCourseNum) {
        super (theTitle, theDept);
        courseNum = theCourseNum;
    }
    public int getNumber() {
        return courseNum;
    }
}
public class AFreshmanSeminar extends ACourse {
  public AFreshmanSeminar (String theTitle, String theDept) {
    super (theTitle, theDept);
  }
  public int getNumber() {
    return SEMINAR_NUMBER;
  }
}
public class AFreshmanSeminar extends ACourse {
    public AFreshmanSeminar (String theTitle, String theDept) {
        super (theTitle, theDept);
        title = theTitle;
    }
    public AFreshmanSeminar () {
    }

    public int getNumber() {
        return SEMINAR_NUMBER;
    }
}

new AFreshmanSeminar ("Lego Robots", "COMP");
new AFreshmanSeminar();
Multiple Constructors

```java
public abstract class ACourse implements Course {
    String title, dept;
    final String DEFAULT_DEPT = "COMP";
    final String DEFAULT_TITLE = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:" + dept + " Number: " + getNumber());
    }
    public ACourse () {
        dept = DEFAULT_DEPT;
        title = DEFAULT_TITLE;
        System.out.println("New course created: " + "Title:" + title + " Dept:" + dept + " Number: " + getNumber());
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}
```
public abstract class ACourse implements Course {
    String title, dept;
    final String DEFAULT_DEPT = "COMP";
    final String DEFAULT_TITLE = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:" + dept + " Number: " + getNumber());
    }
    public ACourse () {
        dept = DEFAULT_DEPT;
        title = DEFAULT_TITLE;
        System.out.println("New course created: " + "Title:" + title + " Dept:" + dept + " Number: " + getNumber());
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}

How to remove code duplication?
CONSTRUCTOR CALLING CONSTRUCTOR

```java
public abstract class ACourse implements Course {
    String title, dept;
    final String DEFAULT_DEPT = "COMP";
    final String DEFAULT_TITLE = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:" + dept + " Number: " + getNumber());
    }
    public ACourse () {
        this (DEFAULT_DEPT, DEFAULT_TITLE );
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}
```

**this** implies constructor of same class.
public abstract class ACourse implements Course {
    String title, dept;
    final String DEFAULT_DEPT = "COMP";
    final String DEFAULT_TITLE = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        super();
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:"+ dept + " Number: " + getNumber());
    }
    public ACourse () {
        this (DEFAULT_DEPT, DEFAULT_TITLE);
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}
CONSTRUCTOR CALLING CONSTRUCTOR

;  
public abstract class ACourse implements Course {
  String title, dept;
  final String DEFAULT_DEPT = "COMP";
  final String DEFAULT_TITLE = "Topics in Computer Science";
  public ACourse (String theTitle, String theDept) {
    title = theTitle;
    dept = theDept;
    System.out.println("New course created: " + "Title:" + title + " Dept:"+ dept + " Number: " + getNumber());
  }
  public ACourse () {
    this (DEFAULT_DEPT, DEFAULT_TITLE );
  }
  public String getTitle() {return title;}
  public String getDepartment() {return dept;}
  abstract public int getNumber();
}

Java complains that instance vars cannot be accessed.
MULTIPLE CONSTRUCTORS

```java
public abstract class ACourse implements Course {
    String title, dept;
    final String DEFAULT_DEPT = "COMP";
    final String DEFAULT_TITLE = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:" + dept + " Number: " + getNumber());
    }

    public ACourse () {
        dept = DEFAULT_DEPT;
        title = DEFAULT_TITLE;
        System.out.println("New course created: " + "Title:" + title + " Dept:" + dept + " Number: " + getNumber());
    }

    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}
```
public abstract class ACourse implements Course {
    String title, dept;
    final String DEFAULT_DEPT = "COMP";
    final String DEFAULT_TITLE = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:"+ dept + " Number: " + getNumber() );
    }
    public ACourse () {
        dept = DEFAULT_DEPT;
        title = DEFAULT_TITLE;
        System.out.println("New course created: " + "Title:" + title + " Dept:"+ dept + " Number: " + getNumber() );
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}

How to remove code duplication?
public abstract class ACourse implements Course {
    String title, dept;
    final String DEFAULT_DEPT = "COMP";
    final String DEFAULT_TITLE = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:" + dept + " Number: "+ getNumber());
    }
    public ACourse () {
        this (DEFAULT_DEPT, DEFAULT_TITLE);
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}
public abstract class ACourse implements Course {
    String title, dept;
    final String DEFAULT_DEPT = "COMP";
    final String DEFAULT_TITLE = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        super();
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: Title:" + title + " Dept:" + dept + " Number: " + getNumber());
    }
    public ACourse () {
        this (DEFAULT_DEPT, DEFAULT_TITLE);
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}
CONSTRUCTOR CALLING CONSTRUCTOR

```java
public abstract class ACourse implements Course {
    String title, dept;
    final String DEFAULT_DEPT = "COMP";
    final String DEFAULT_TITLE = "Topics in Computer Science";
    public ACourse (String theTitle, String theDept) {
        title = theTitle;
        dept = theDept;
        System.out.println("New course created: " + "Title:" + title + " Dept:"+ dept + " Number: " + getNumber());
    }
    public ACourse () {
        this (DEFAULT_DEPT, DEFAULT_TITLE );
    }
    public String getTitle() {return title;}
    public String getDepartment() {return dept;}
    abstract public int getNumber();
}
```

Java complains that instance vars cannot be accessed.
public abstract class ACourse {
    String title, dept;
    
    public ACourse (String theTitle, String theDept) {
        super();
        title = theTitle;
        dept = theDept;
    }

    public String getTitle() {
        return title;
    }

    public String getDepartment() {
        return dept;
    }

    abstract public int getNumber();
}
Abstract method that returns an instance of some type T – like a factory, it creates and initializes an object.
INIT METHODS

- Allows initialization after object is created.
- Initializer can be different from creator
- Init methods can be in interfaces
- Init method(s) recommended but not required
Visibility of SuperClass Members

Should non public identifiers be visible in subclasses?

ACourse

getDepartment()

getNumber()

AFreshmanSeminar

getTitle()

getNumber()

ACourse

title
depth

ARegularCourse

courseNum

getNumber()
ACCESS CONTROL ON VARIABLES AND INHERITANCE, PACKAGES

- **public**: accessible in all classes.
- **protected**: accessible in all subclasses of its class and all classes in its package.
- **default**: accessible in all classes in its package.
- **private**: accessible only in its class.
public class ARegularCourse extends ACourse {
    int courseNum;
    public ARegularCourse (String theTitle, String theDept, int theCourseNum) {
        super (theTitle, theDept);
        courseNum = theCourseNum;
        title = theTitle;
    }
    public int getNumber() {
        return courseNum;
    }
}