"A class, in Java, is where we teach objects how to behave."
-- R. Pattis

Section 11: Object-oriented design

Objectives:

- Constructors
- Array fields in classes
- Classes versus instances
- Class design

Historical note …

- The Xerox Palo Alto Research Center (PARC), founded in 1970, is at the origin of many important contributions:
  - The first workstation (Alto) with graphical user interface (GUI, with windows and icons) and mouse
  - The first text editor WYSIWYG
  - The InterPress language (predecessor of PostScript) for describing pages to be printed
  - The Ethernet protocol for local area networks
  - The Smalltalk object-oriented programming language, with graphical development environment (designed by Alan Kay)
  - The laser printer
  - …

Object-orientation

- The approach we have taken with our student class is an "object oriented" approach:
  - We have a class that is a template for the creation of objects.
    - Student objects can be referred to as INSTANCES of the CLASS Student.
  - Object instances have instance methods that use the field values for a specific object
    - e.g. getFinalMark( ) will have different results for different objects because this.exam is different for different objects
  - If you want the object to do something for you, you have to ask it by calling a method on that object.
    - That is, you can't sneak inside an object from outside the class and change the private field values.
  - You also can't call an instance method, without using an object reference:

    x ← 3.0 + getFinalMark( ) is meaningless. Whose final mark are we referring to?

341

Initialization of Objects

- When we create a new Student, we will usually want to provide values for all the fields in the object.

```
aStudent = new Student( );
aStudent.id = 1234567;
aStudent.midterm = 60.0;
aStudent.exam = 80.0;
aStudent.forCredit = true;
```

- A special kind of method called a CONSTRUCTOR, can be used to initialize values inside an object as the object is being created.

```
aStudent = new Student( 1234567, 60,0, 80.0, true);
```

342

Constructors

- A constructor is a special method in a class used to create an object.
  - the name of the method is the same as the class;
  - no return type
  - usually public;
  - may or may not have parameters.
- The parameters, if any, in a constructor are used to initialize the values of the object.
- Because there may be different ways to initialize an object, a class may have any number of constructors, distinguished from each other by different parameter lists.

343

Implementation in Java

- The following is a constructor that sets a value for all of the fields in the **Student**:

```
class Student
{
   // ... fields would be defined here ...
   public Student(int theId, double theMidterm, double theExam, boolean
                  isForCredit)
   {
      this.id = theId;
      this.midterm = theMidterm;
      this.exam = theExam;
      this.forCredit = isForCredit;
   }
   // ... Other methods ...
}
```

- This constructor could be used as follows:

```
Student aStudent = new Student(1234567, 60.0, 80.0, true);
```

344

Constructors of class Student

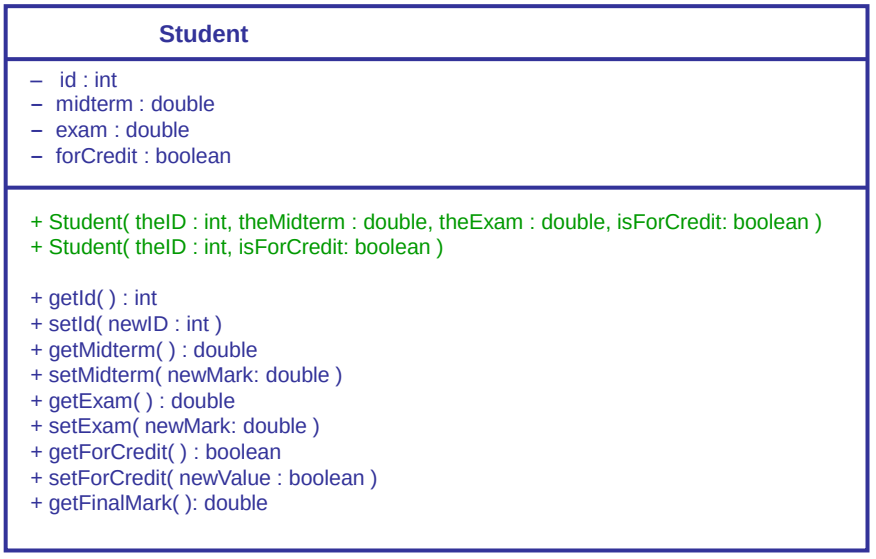
- If we are doing course registrations, we may only want to provide the ID number and whether the student is taking the course for credit. (We don't know the student's marks yet!)
- We could also provide the following constructor:

+ Student(theID : int, isForCredit : boolean) *UML*

```
public Student(int theID, boolean isForCredit )
{
   this.id = theID;
   this.midterm = 0.0;            // a "safe" value
   this.exam = 0.0;               // a "safe" value
   this.forCredit = isForCredit;
}
```

- When there is more than one constructor, they must have parameter lists that can be distinguished by the number, order, and type of parameters.

Add Constructors to the Class



Constructors of class Student

- Here is a constructor with no parameters:

  + Student( )

```
public Student( )
{
   this.id = 0;
   this.midterm = 0.0;
   this.exam = 0.0 ;
   thisl forCredit = false;
}
```

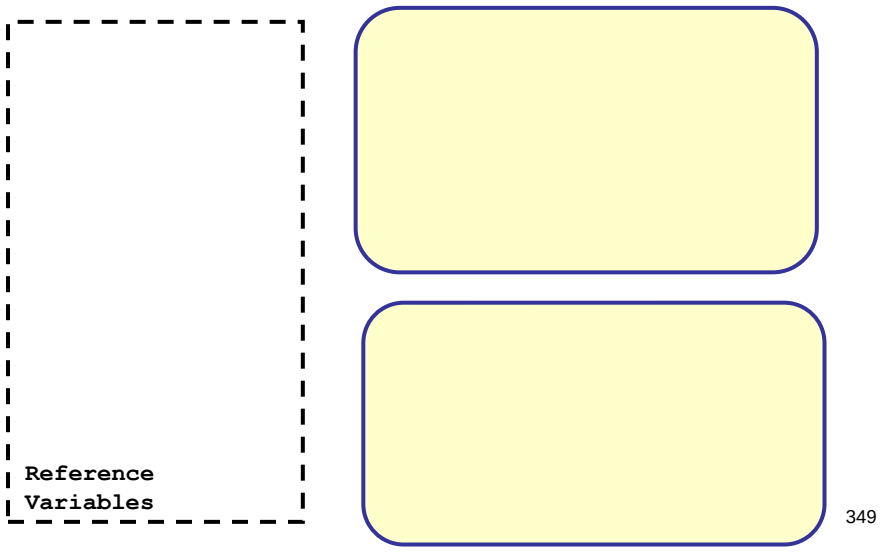
- A constructor without parameters is called a default constructor.
  - It is recommended to always define a default constructor that sets every field to a "safe" value.
- If, and only if, a class does not define any constructor, the Java compiler invisibly creates a default constructor that does nothing.

Exercise 11-1: Using Constructors

- **Show how objects are created and referenced by the following main method.**

```
public class Section11
{
   public static void main(String [] args)
   {
        Student aStudent; // reference variable
        Student meToo;    // another reference variable
        Student bStudent; // a third reference variable
                  •
                  •
        aStudent = new Student(1234567,60.0,80.0,true);
        meToo = new Student(7654321,true);
        bStudent = aStudent;
          •
          •
          •
   }
}
```

Exercise 11-1: Using Constructors



349

Array Fields in Classes

Student
– id : int – midterm : double – exam : double – forCredit : boolean – assignments : double[ ]
+ Student(theID : int, theMidterm : double, theExam : double, isForCredit: boolean) + Student(theID : int, isForCredit: boolean) + getId( ) : int + setId(newID : int) + getMidterm( ) : double + setMidterm(newMark: double) + getExam( ) : double + setExam(newMark: double) + getForCredit( ) : boolean + setForCredit(newValue : boolean) + getFinalMark( ): double

- A field of a class may have any type. In particular, a class may have a field of an array type (reference variable).
- Add an array of double to class Student representing assignment marks:
- Remember, arrays are not created automatically! The array assignments will have to be created after a Student object is created.

350

Array Fields in Classes

```
public class Student
{
   private int id ;
   private double midterm ;
   private double exam ;
   private boolean forCredit;
   private double[] assignments;

   // methods
}
```

- The array reference variable **assignments** is still **null**.

Array field initialization

- Here is a constructor that creates and initializes an array in an object. The constructor has a parameter that is the number of assignments.

```
public Student( int numberOfAssignments )
{
   this.id = 0;
   this.midterm = 0.0;
   this.exam = 0.0 ;
   this.forCredit = false;
   this.assignments = new double[numberOfAssignments];

   // loop to initialize each item in array
   int index;
   for ( index=0; index < numberOfAssignments; index = index+1 )
   {
      this.assignments[index] = 0.0;
   }
}
```

Accessors for an Array Field

- An accessor for an array field could:
  - Return a reference to the entire array

    + getAssignments() : double[ ] *UML*

```
public double [] getAssignments()
{
   return assignments;
}
```

  - Return one of the values in the array, with an extra parameter to select the array index.

    + getAssignment (assignNumber: int) : double *UML*

```
public double getAssignments( int assignNumber )
{
   return assignments[assignNumber];
}
```

- Which approach is better?

353

Exercise 11-2: Calculation of the Final Mark ?

- Write Java methods **`calcAssignAvg( )`** and **`getFinalMark( )`** for our **`Student`** class that returns a **`double`** with a student's final mark, where:
  - The final mark is 55% of the exam, plus 20% of the midterm, plus 25% of the average of 5 assignments.

```
public double calcAssignAvg()
{
```

354

```
}
```

Exercise 11-2: Calculation of the Final Mark

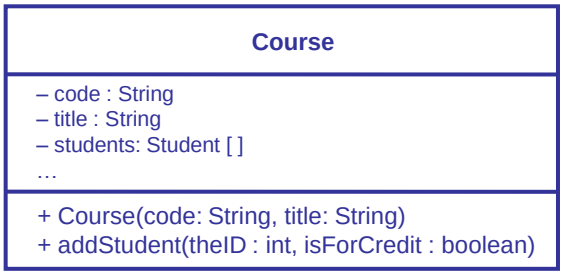
- Write Java methods `calcAssignAvg( )` and `getFinalMark( )` for our `Student` class that returns a `double` with a student's final mark, where:
  - The final mark is 55% of the exam, plus 20% of the midterm, plus 25% of the average of 5 assignments.

```
public double getFinalMark()
{



}
```

355

Course information

- Now that we have a class that stores information about one Student, how can we use this create a class Course that stores information about all students?



356

Exercise 11-3: Arrays of objects

- Show how objects are created and referenced by the following **main** method.

```
public class Section11
{
   public static void main(String [] args)
   {
      Course aCourse;
      aCourse = new Course("ITI1120","Intro. to Comp.");
      aCourse.addStudent(123456,true);
      aCourse.addStudent(654321,false);
   }
}
```

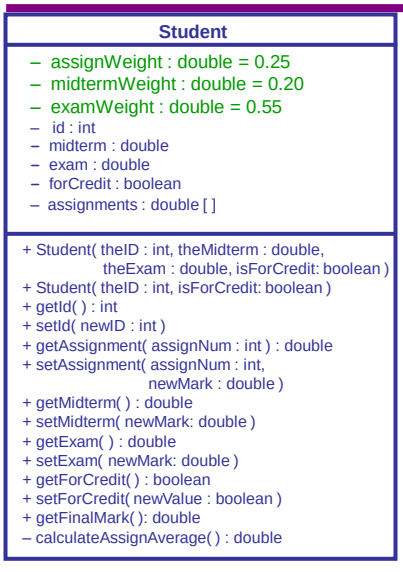

Exercise 11-3: Array of Student Objects ?

Common values for a Class

- For our student and course objects, we have been using instance variables, and creating instance methods.
  - There is a set of instance variables created for each new object.
  - The instance methods use the instance variables for the object on which they are called.

- Suppose we have a value that we want every student object to know about.
  - Examples: The weights of the final exam, midterm, and assignments for calculating a student's final mark.

Adding weights as attributes



- The problem with this approach is that EVERY Student object will have copies of assignWeight, midtermWeight, and examWeight, which takes up extra storage for each student
  - How much extra storage would this take if there were one object for every ITI 1120 student?

- If the weights change, every object would have to be updated.

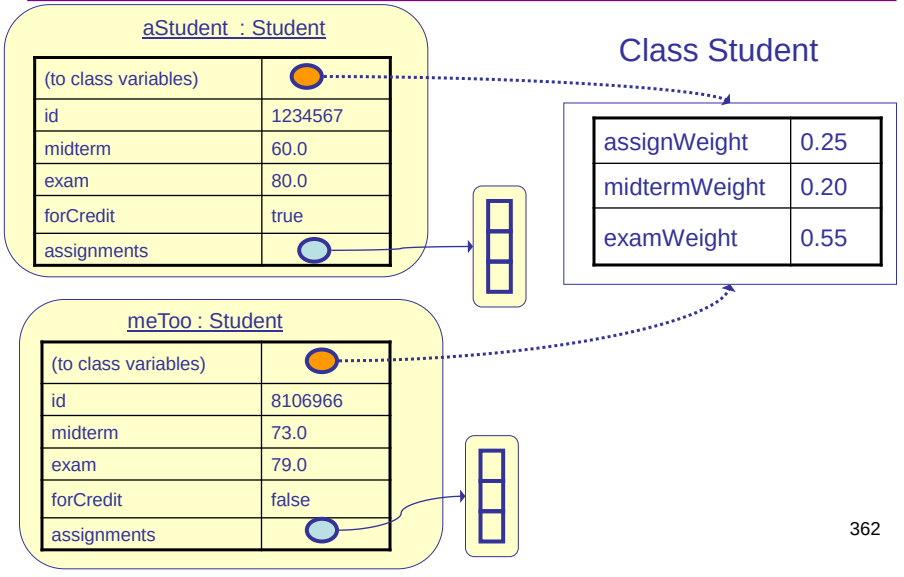
- We want the weights to be consistent among all Student objects.

Class Variables

- Another type of variable we can have with a class is a "class variable" (also known as "static variable ").
- Class variables are NOT stored inside individual objects, because they belong to the entire class.
  - Instead, they are stored with the class.
- EVERY object created from the class has access to the class variables, even if they are private.
  - If class variables are public, then methods outside the class also have access to the class variables.

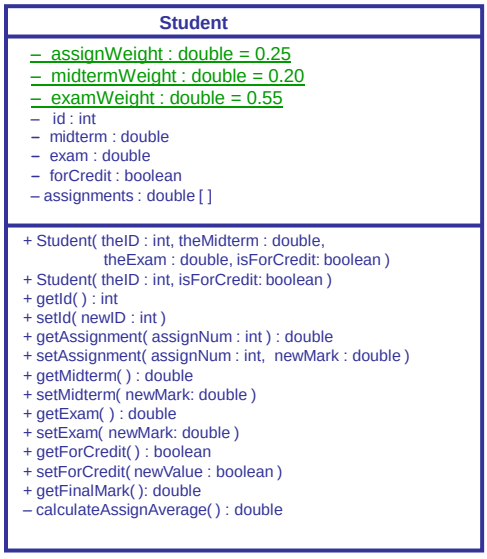
361

Class and Instance variables



362

Class variables in UML diagrams



- Class variables are underlined; instance variables are not.
- Note that the initial values of the class variables has been specified.

Translation to Java

```
public class Student
{
   // Class variables                        (applies to all students)
   static private double assignWeight = 0.25;
   static private double midtermWeight = 0.20;
   static private double examWeight = 0.55;

   // Instance variables                     (one copy per student object)
   private int id ;
   private double midterm ;
   private double exam ;
   private boolean forCredit;
   private double[] assignments;

   public double getFinalMark()
   {
      double assignAvg = this.calculateAssignAverage( );
      double finalMark = Student.midtermWeight * this.midterm
             + Student.examWeight * this.exam
             + Student.assignWeight * this.assignAvg;
      return finalMark;
   }
}
```

Class Methods

- Now we can change the final mark calculation for all students by changing the value of the weights.
- We want to write a modifier method for each weight.
  - This modifier should be a CLASS method, because the values are associated with the class instead of the objects.
  - We can also set the weights before creating any student objects.
- A class method is called as follows:

  ClassName . aMethodName( )
  - Just like the **Math** class!
- A class method CANNOT use any instance variables, because it is not associated with any particular object.

Class methods in UML diagrams

- Class methods are underlined; instance methods are not.

`static` methods in Java

- As with class variables, class methods are indicated by the keyword **`static`**.
  - Note that **`main`** is always a class method.
- Write Java class methods for **`Student`** to set the values of the weights for the assignments, midterm, and final exam

```
public static void setExamWeight( double newWeight )
{
   Student.examWeight = newWeight ;
}
public static void setMidtermWeight( double newWeight )
{
   Student.midtermWeight = newWeight ;
}
public static void setAssignmentWeight( double newWeight )
{
   Student.assignWeight = newWeight ;
}
```

367

Exercise 11-4 – Using Class Variables and Methods

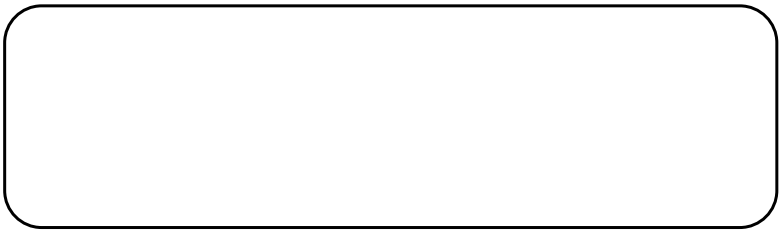
- What output would be produced by the following `main` method.

```
public class Section11
{
   public static void main(String [] args)
   {
       int anum;
       Student aStudent; // reference variable
       Student meToo;    // another reference variable
       aStudent = new Student(1234567,60.0,80.0,true);
       meToo = new Student(7654321,54.5, 83.4, true);
       for(anum=0 ; anum<5 , anum=i+1)
       {
          aStudent.setAssignment(anum, 60.0);
          meToo.setAssignment(anum, 65.0);
       }
       System.out.println("The mark for student "+aStudent.getId()+
                          " is "+ aStudent.getFinalMark();
      Student.setMidWeight(0.30);
      Student.setAssignmentWeight(0.15);
      System.out.println("The mark for student "+meToo.getId()+
                          " is "+ meToo.getFinalMark();
       System.out.println("The mark for student "+aStudent.getId()+
                          " is "+ aStudent.getFinalMark();
    }
}
```

368

Exercise 11-4 – Using Class Variables and Methods

Terminal Window



Summary of Class Design (1)

- In an object-oriented language such as Java, designing a class is a large part of the effort to create software.

"Classes struggle, some classes triumph, others are eliminated." --Mao Zedong

- Decisions have to be made as to:
  - What information should be in the class?
    - What fields should each object have?
    - What fields should be associated with the class?
    - What type are the fields?
    - How do we initialize, set, and change the fields?
  - What are the operations we may want to ask the class to perform?
    - What other instance methods are needed?
    - What other class methods are needed?
    - What are the algorithms for all of these methods?

Summary of Class Design (2)

- Some things to keep in mind when making class design decisions:
  - How might the class be modified in the future?
    - Is there anything in the class that is "hard coded" that perhaps should be a variable?
  - Safety:
    - Is there a chance that a variable is used before it receives a value?
    - When a method is called, what does the method assume about the parameter values? Are those assumptions checked?

Example of a Class Design: Fraction

- We shall define class from which objects represent objects such as 2/3, (-1)/5, or 7/4.
- Since 2 / 3 = 4 / 6 = 6 / 9, and (-1) / 2 = 1 / (-2), etc., fractions with different numerators and denominators can still represent the same fraction.
- We shall store fractions in a « standard form »:
  1. The numerator and denominator do not have any common factor other than 1 or -1.
  2. The denominator must be positive.
- In math, the denominator of a fraction can never be 0. But we may not prevent users from creating a fraction with a 0 denominator. Java has a mechanism called **exception handling** which handles error conditions (such as divide by 0). We shall NOT study exception handling in this course.

Specification for a Fraction class

- A fraction consists of a numerator and a denominator.
- The numerator of a fraction is an integer.
- The denominator of a fraction is an integer not 0.
  - If the denominator is not specified at creation, it is assumed to be 1.
- A fraction is always in "standard form"; that is
  - The greatest common divisor (GCD) of the numerator and denominator is always 1
  - The denominator is always positive.
    - Example: 6/-9 should be represented as -2/3
    - Special case: if the numerator is 0, the fraction is represented as 0/1
- A fraction with denominator 1 should be displayed as the equivalent integer; otherwise in the form numerator/denominator.

Exercise 11-5: Designing a Fraction class ?

- What information do we need to store in a Fraction?

- What operations do we need?
  - [Aside from creating fractions, the only mathematical operation we will implement is addition of two fractions]

Exercise 11-6: Simplify Fraction to Standard Form

- To make sure that each `Fraction` instance will be in lowest terms, a method `simplify` will be used.
- Assume that you have a method `gcd(a,b)` that will return the greatest common divisor of two integers.
- Write Java methods to put a fraction into standard form.

Exercise 11-7: Method for GCD

- A recursive GCD algorithm for gcd(a,b):
  - If a mod b is 0, gcd(a, b) is b
    - a mod b is the remainder when a is divided by b
  - Otherwise, gcd(a,b) is gcd(b, a mod b)
- Question: will this algorithm always reach the base case?
  - Note that a mod b is at most b – 1.
- Careful: what if b is set to 0?

Exercise 11-8: Fraction Constructors ?

- Write constructors for a `Fraction` that:
  - take 2 integers: the numerator and the denominator
  - takes 1 integer, representing an integer that is to be converted to a Fraction

Exercise 11-9: Displaying Fractions ?

- Write a Java method to display a `Fraction`.
- Sample usage:
  - `Fraction f1 = new Fraction ( 6, -9 );`
  - `f1.display( );`
- Result: `-2/3`

Exercise 11-10: Adding Fractions

- Write a Java method that will add two Fractions.
  - Sample usage:

```
Fraction f1 = new Fraction( 1, 2 );
Fraction f2 = new Fraction( 1, 3 );
Fraction sum = f1.addTo( f2 );
sum.display( );
```

  - Result: `5/6`

Exercise 11-11: Adding an Integer to a Fraction

- Write a method that will and an integer to a Fraction:

```
Fraction f1 = new Fraction( 5, 2 );
Fraction sum = f1.plus( 3 );
sum.display( );
```

  - Result: `11/2`

Other Arithmetic Operations

- As an additional home exercise, try to create similar methods for other operations (subtraction, multiplication and division) on two fractions and with a fraction and integer.
- What is special about division?

Some final words…

- "At the source of every error which is blamed on the computer, you will find at least two human errors, one of which is the error of blaming it on the computer. "
  - Anonymous
- "We shall do a much better programming job, provided we approach the task with a full appreciation of its tremendous difficulty, provided that we respect the intrinsic limitations of the human mind and approach the task as very humble programmers. "
  - Alan Turing