ITI 1121
Introduction to Computing II

Lecture 03-
2016

These slides are based on :

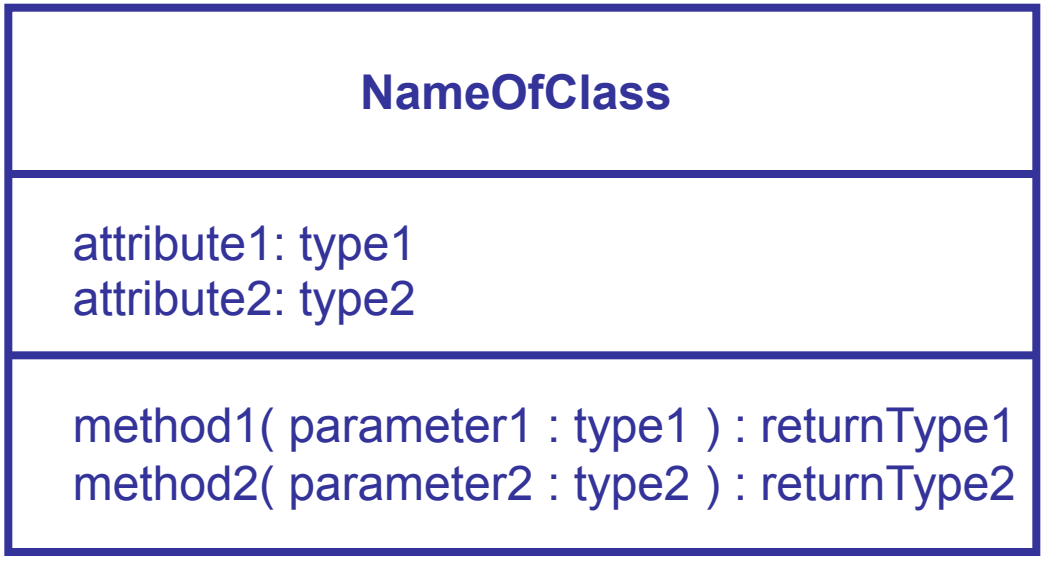
ITI1120/ITI1121 slides by Profs: D. Inkpen/M. Turcotte

Agenda

1. Java programs
2. What is a variable?
3. What is a data types?
    1. Primitive data types.
    2. Reference data types
        1. Example 1:Arrays
        2. Example 2: Strings
4. Methods in Java
5. **Classes in Java**

3rd data type: Classes

Class diagram



"Attributes" are like the field variables in a record

- This form of diagram is from a notation called the "Unified Modelling Language" or UML

Classes and Objects

- A "class" can be used as a template to create objects with identical sets of attributes.

- The class can also contain methods (algorithm models) to perform calculations on the attributes of objects created from the class (and/or external data).

Classes versus Objects versus

- **A Class is the templates to create of objects of that particular class**

We use the class student to create as many student objects as we want....

- **You cannot design an object, you are designing a class that species the characteristics of a collection of objects. The class then serves to create the instances**

- **Are instance and object two deferent concepts?**

No. Instance and object refer to the same concept, the word instance is used in sentences of the of the form \the instance of the class . . . ", when talking about the role of an object.

.

Naming classes

Use **singular nouns** whose **first letter is capitalized**.

Conventions are very important to make the programs more readable.

The following declaration,

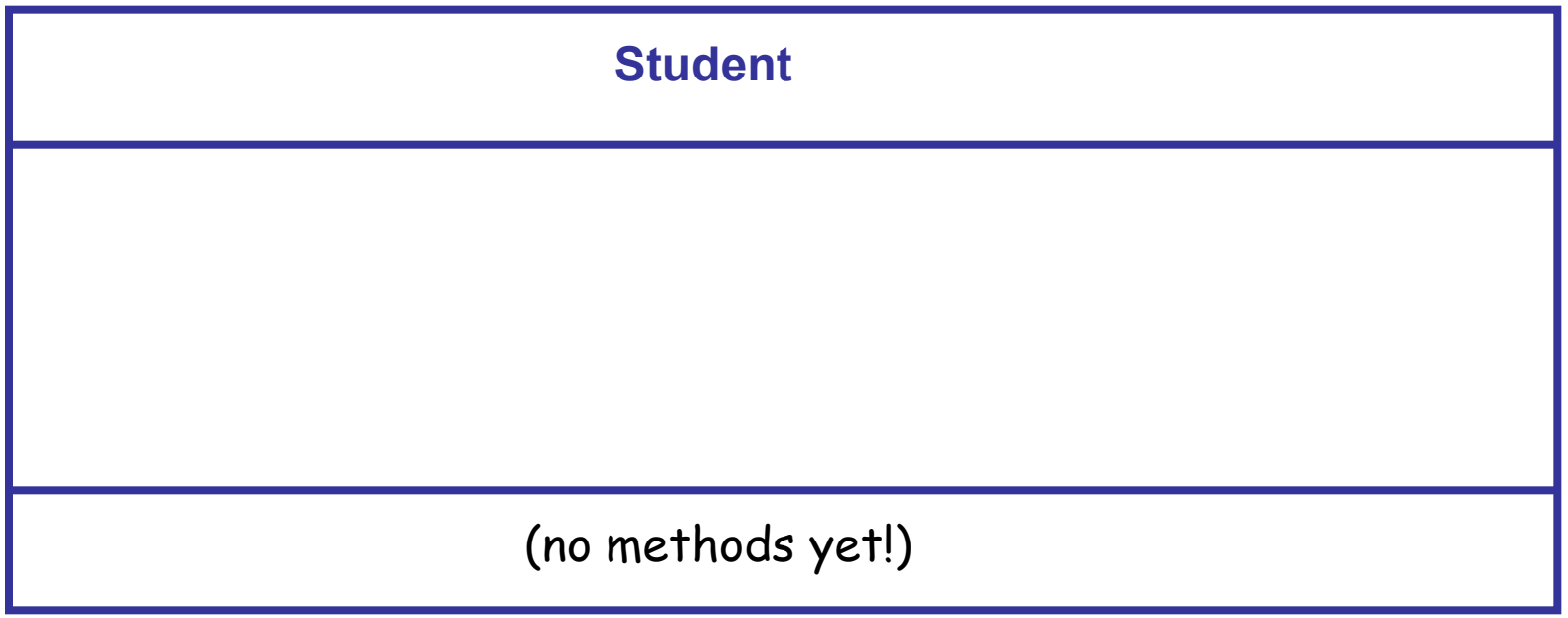
```
Counter counter;
```

clearly indicates a reference variable, **counter**, to be used to designate an instance of the class **Counter**.

A practical example is **System.out.println("hello")**.

An Example

- For each student, we want to store their ID number, their midterm score, their exam score, and whether or not the student is taking the course for credit. *We shall deal with the final mark later.*



Student Information

- How can we store a all the information about each student in a course?
  - ID (student number) (integer)
  - midterm mark (real)
  - final exam mark (real)
  - is taking this course for credit (Boolean)

- What is the problem with the following solutions:
  - Each value is stored in a separate variable:

    Difficult to manipulate / exchange all information about a student.
  - Put all the values into an array:

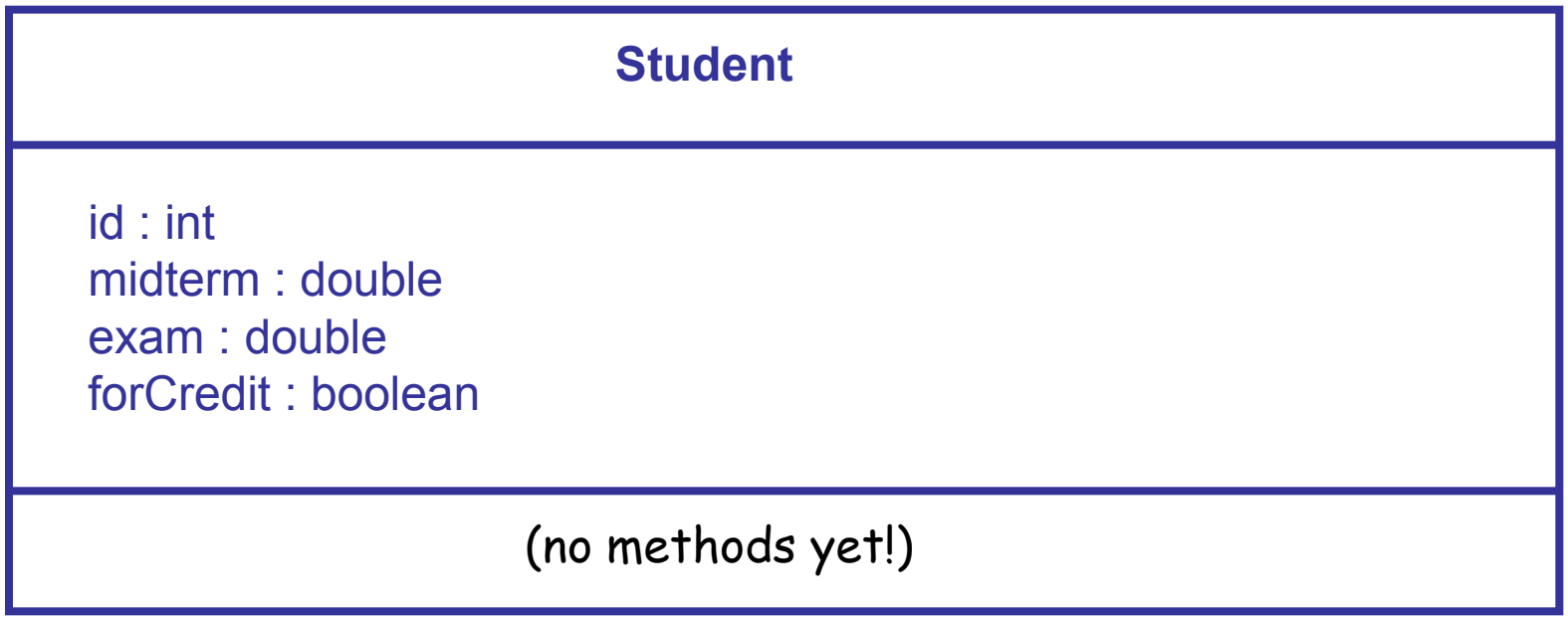
    The variables are not of the same type.

Translation to Java

```
public class Student
{
   // variables




   // methods
}
// In your main program ..declare aStudent
  reference Variable

// Create a Student object referenced by aStudent
```

First version of a Student Class

- For each student, we want to store their ID number, their midterm score, their exam score, and whether or not the student is taking the course for credit. *We shall deal with the final mark later.*



Translation to Java

```
public class Student
{
   public int id;
   public double midterm;
   public double exam;
   public boolean forCredit;
   // methods
}
```

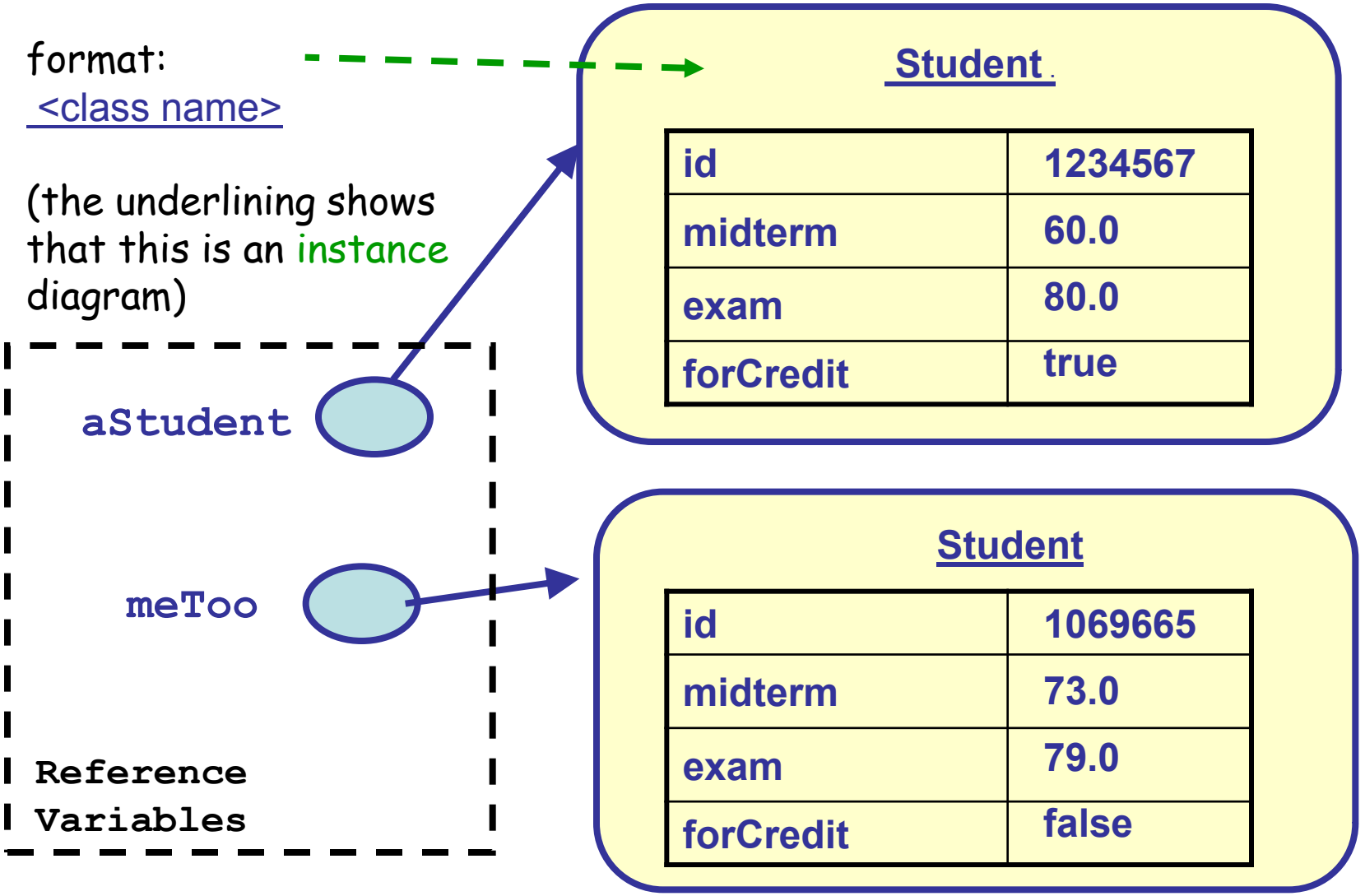
How to use the class

```
   // In main: Declare aStudent reference Variable
Student aStudent;           // declare reference variable

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

Student meToo;
meToo = new Student();
meToo.id = 81069665;
meToo.midterm = 73.0;
meToo.exam = 77.0;
meToo.forCredit = false;
```

How to use the class

format:
<class name>

(the underlining shows that this is an instance diagram)



Information hiding...Step 1

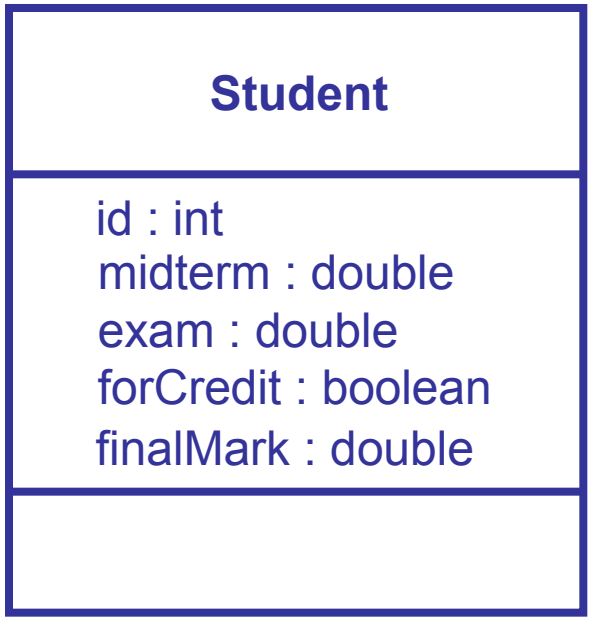
- Suppose we want to modify the Student class to keep the course final mark, which is 20% of the midterm mark plus 80% of the final mark.
  - We could add a field finalMark to our class.

- We want to make sure that

  finalMark = (0.2 midterm + 0.8 exam)

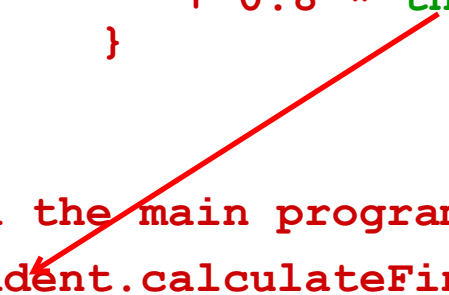
  is always true for consistency.

Example: Student class- adding variables



Translation to Java

```
public class Student
{
    public int id;
    public double midterm;
    public double exam;
    public boolean forCredit;
    public double finalMark
    // methods
     Public void calculateFinalMark()
          {
              this.finalMark = 0.2 * this.midterm
              + 0.8 * this.exam;
          }

}
//In the main program
aStudent.calculateFinalMark();
```



Where did `this` come from?

- `this` refers to the called object.
- During the call `aStudent.calculateFinalMark()` `this` is a reference to `aStudent`
  - … and so `this.forCredit` is translated to `aStudent.forCredit`, which is is a boolean variable with "true" as its value
  - .
- During the call `meToo.calculateFinalMark()`, `this` is a reference to `meToo`
  - … and so `this.forCredit` is `meToo.forCredit`, which is a Boolean variable with "false" as its value

Where did `this` come from?

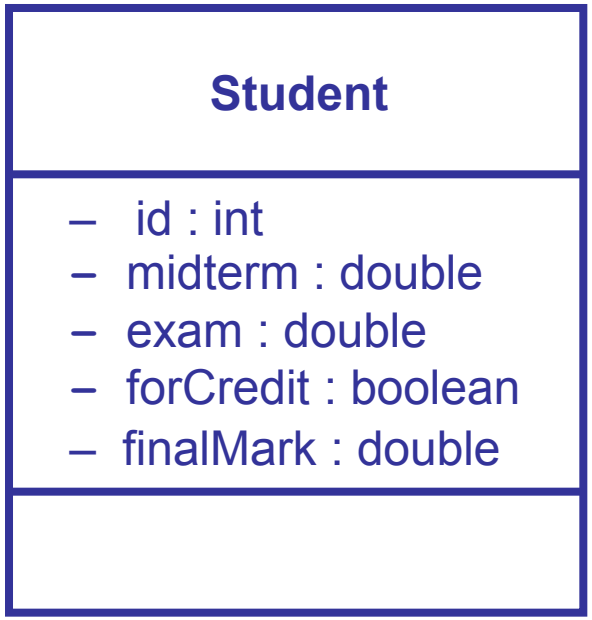
- In most cases, we don't actually have to use `this` to refer to the object on which a method is called.
  - Inside the Student class:
    - `finalMark`can be used instead of `this.finalMark`.

    As we will see later on this is moslty used to remove ambiguity.. When it is not clear which variable or object you are referring to.

Information hiding.....

- Now suppose we want to make sure that only the finalMark to be changed only if the midterm and the final marks are changed? And we don't want to allow the main program to change it directly through the dot operator **`aStudent.finalMark=100`**

Example: Student class- hiding variables



- The – in front of the variable indicates that the attribute is private.
- By declaring a field to be private, only methods declared inside the class are allowed access to the field value (either for viewing the value, or changing the value).

Creating the Student Class

```
public class Student
{
    // were previously public
    private int id;
    private double midterm;
    private double exam;
    private boolean forCredit;
    private double finalMark;
   // methods
}
```

How do we use the class

- If we try the following:

```
Student aStudent = new Student();
aStudent.finaMark = 100 ; // error!
```

the compiler returns an error since access to finalMark is no longer allowed from outside the class.

Information Hiding

- To ensure independence relative to other parts of your program (which helps reduce the effort of maintenance), fields are (almost) always **`private.`**
- This information hiding is also called data abstraction and also encapsulation).

- The private fields and methods cannot be accessed directly but only from methods in the class.

- If (and only if) necessary, can define a few public methods to allow other parts of the program to access fields.

- The public methods represent the interface of the class relative to other parts of the program.

But how to provide access?

- But now no one can access id or any other variable either to set it or to view it... Which is not a very good idea...

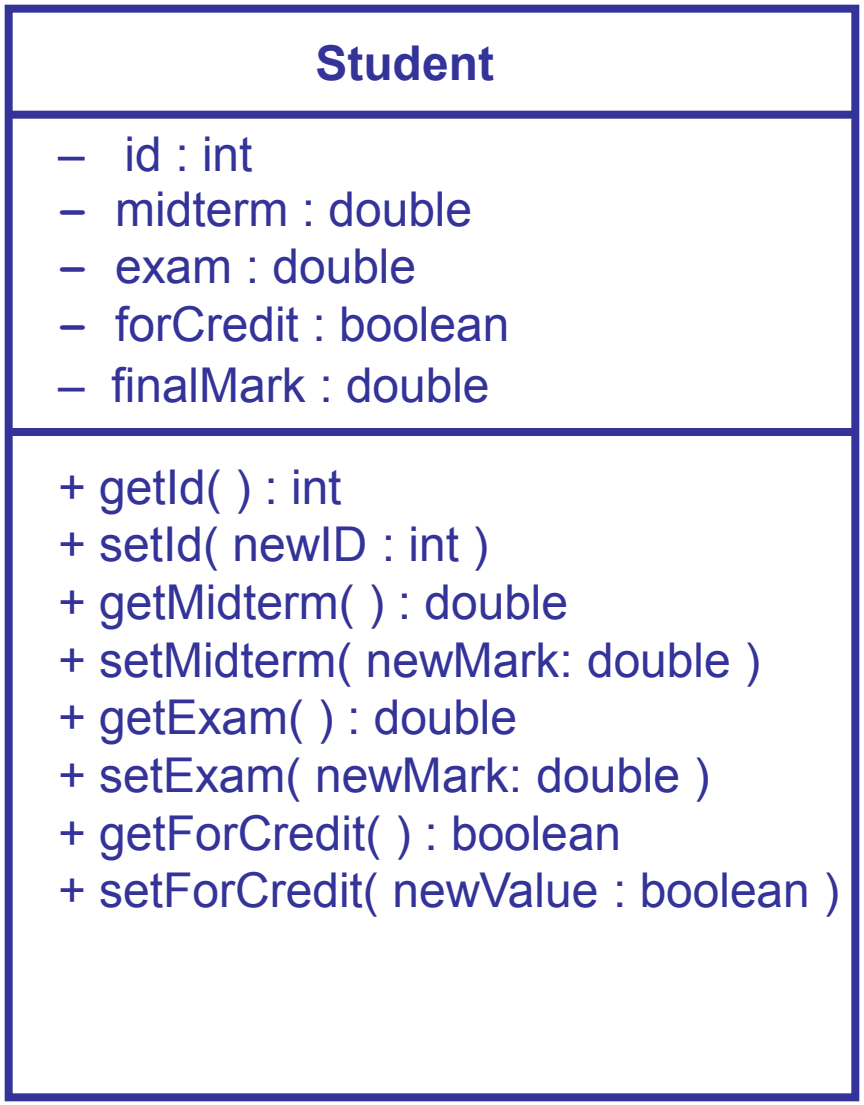
- we can create additional access methods in the class **`Student`**:
  - "accessor": requests to see the value of a private field.
  - "modifier": requests to modify the value of a private field.

Accessors and Modifiers

- Accessor
  - A public instance method (called using a reference to an object);
  - Returns the value of the field of the object;
  - Has no parameters;
  - Often called `getFieldName` (also called a getter method).

- Modifier
  - A public instance method (called using a reference to an object);
  - Assigns a value to a field;
  - Accepts values in a parameter of the same type as the field;
  - Often called `setFieldName` (also called setter method).

Student class with Information Hiding



Accessors and Modifiers

- Examples for the forCredit field in the class:

boolean getForCredit( )

  - method to return the value of forCredit
  - the + indicates that the method has public visibility
  - the return type is boolean, and in UML notation, appears at the end of the method.

void setForCredit(newValue : boolean)

  - method to change the value of forCredit
  - one parameter newValue, of type boolean
  - no return value

Translation to Java

```
public class Student
{
   // Attributes

   private int id;
   private double midterm;
   private double exam;
   private boolean forCredit;
   private double finalMark;

   // Methods

   public int getId()
   {
      // insert code here
   }
   public void setId( int newId )
   {
      // insert code here
   }
   public double getMidterm()
   {
      // insert code here
   }
   public void setMidterm( double newMark )
   {
      // insert code here
   }
   // continued at right
```

```
   // continued from left side

   public double getExam()
   {
      // insert code here
   }
   public void setExam( double newMark )
   {
      // insert code here
   }
   public boolean getForCredit()
   {
      // insert code here
   }
   public void setForCredit( boolean newValue )
   {
      // insert code here
   }
   public double getFinalMark()
   {
      // insert code here
   }
.......


} // end of class Student
```

Implementing Java accessors and modifiers

```
public class Student   // not all attributes/methods shown!
{
    // attribute
    boolean forCredit;
    // … other attributes declared

    // accessor:  return the requested value
    public boolean getForCredit()
    {
        return this.forCredit ;
    }

    // modifier: save the requested value in object's
     // attribute

    public void setForCredit( boolean newValue )
    {
        this.forCredit = newValue;
    }
    // …other methods are similar
}
```

Where did `this` come from?

- When the fields of our Student class were public, we distinguished between the same field in two record objects with the variable name and the dot operator:
  - `aStudent.forCredit` versus `meToo.forCredit`
- Likewise, when a method inside the class wants to work with "the value of the field for the object on which I was called", `this` refers to the called object.
- During the call `aStudent.getForCredit()`, `this` is a reference to `aStudent`
  - … and so `this.forCredit` is `aStudent.forCredit`, which is true.
- During the call `meToo.getForCredit()`, `this` is a reference to `meToo`
  - … and so `this.forCredit` is `meToo.forCredit`, which is false.

Calling Java Accessor and Modifier Methods

- Again, use the dot operator (.)
- The following code causes errors, why?

```
Student aStudent = new Student();
aStudent.id = 1234567;                  // error here!
int myId = aStudent.id;                 // error here!
System.out.println( myId );
```

- The compiler enforces the private access to `id` .

- Solution: Instead, use the modifier and accessor methods.

```
Student aStudent = new Student();
aStudent.setId( 1234567 );          //ok!
int myId = aStudent.getId( );       //ok!
System.out.println( myId );
```

- Now suppose we want to recalculate the final mark every time we set the midterm or final? But no where else..

Back to Information Hiding

- To implement our strategy of hiding the finalMark field, we can do the following:
  - Set the variable as private
  - We will provide an accessor method for finalMark, but NOT a modifier method.
  - We can provide a method recalculateFinalMark() to recalculate the final mark if the midterm or exam marks are changed.
  - The modifier methods setMidterm() and setExam() will call recalculateFinalMark() so that they automatically update the final mark.
- We should also restrict access to recalculateFinalMark() because it isn't meant for use outside the class.

Implementing Information Hiding

- The following implements our strategy where the final mark can only be changed by modifying the midterm or exam values.

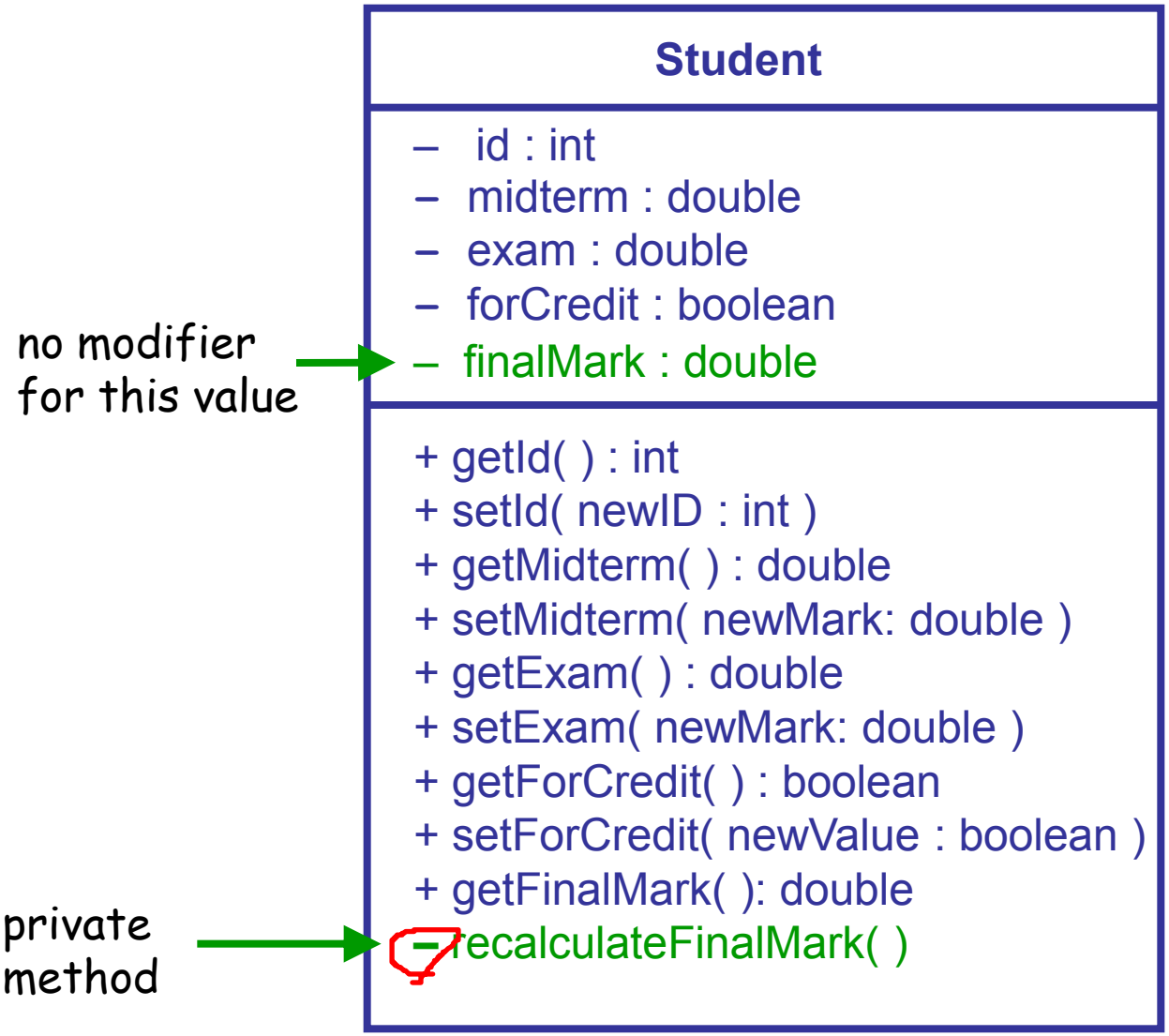
```
public class Student
{
    // attributes and other methods would go here
    public void setMidterm( double newValue )
    {
        this.midterm = newValue;
        this.recalculateFinalMark( );
    }

    public void setExam( double newValue )
    {
        this.exam = newValue;
        this.recalculateFinalMark();
    }

    public void recalculateFinalMark()
    {
        this.finalMark = 0.2 * this.midterm + 0.8 * this.exam;
    }
}
```

Implementing Information Hiding

- The following implements our strategy where the final mark can only be changed by modifying the midterm or exam values.

```
public class Student
{
    // attributes and other methods would go here
    public void setMidterm( double newValue )
    {
        this.midterm = newValue;
        this.recalculateFinalMark( );
    }

    public void setExam( double newValue )
    {
        this.exam = newValue;
        this.recalculateFinalMark();
    }

    private void recalculateFinalMark()
    {
        this.finalMark = 0.2 * this.midterm + 0.8 * this.exam;
    }
}
```

Student class with Information Hiding



Translation to Java

```
public class Student
{
   // Attributes

   private int id;
   private double midterm;
   private double exam;
   private boolean forCredit;
   private double finalMark;

   // Methods

   public int getId()
   {
      // insert code here
   }
   public void setId( int newId )
   {
      // insert code here
   }
   public double getMidterm()
   {
      // insert code here
   }
   public void setMidterm( double newMark )
   {
      // insert code here
   }
   // continued at right
```

```
   // continued from left side

   public double getExam()
   {
      // insert code here
   }
   public void setExam( double newMark )
   {
      // insert code here
   }
   public boolean getForCredit()
   {
      // insert code here
   }
   public void setForCredit( boolean newValue )
   {
      // insert code here
   }
   public double getFinalMark()
   {
      // insert code here
   }

   private void recalculateFinalMark()
   {
      // insert code here
   }
} // end of class Student
```

Translation to Java

- The following implements our strategy where the final mark can only be changed by modifying the midterm or exam values.

```
public class Student
{
   // attributes and other methods would go here
   public void setMidterm( double newValue )
   {
      this.midterm = newValue;
      this.recalculateFinalMark( );
   }

   public void setExam( double newValue )
   {
      this.exam = newValue;
      this.recalculateFinalMark();
   }

   private void recalculateFinalMark()
   {
      this.finalMark = 0.2 * this.midterm + 0.8 * this.exam;
   }
}
```

Benefits of Information Hiding (1)

- One of the most common causes of problems historically has been when all parts of a program have access to all program variables.
  - For example, when someone makes a change to a large program, the new code may make changes to data that some other part of the program assumed would not be modified.

> "Successful software always gets changed." - *F. Brooks*

- With information hiding, we can keep the code better partitioned so that changes will be less likely to cause unwanted side effects.

Benefits of Information Hiding (2)

- We can also make changes inside a class that will not affect users of the class.

- Example: Suppose we decide that the finalMark field really doesn't need to be stored in the Student class.
  - Instead, we can calculate the final mark when anyone asks for it:

```
public double getFinalMark()
{
    return 0.2 * this.midterm + 0.8 * this.exam;
}
```

  - This means we can remove the method recalculateFinalMark( ), and the calls to it in setMidterm( ) and setFinal( ).

- Making these changes will not affect any user of the class:
  - For example, meToo.getFinalMark( ) still behaves as it did before.
  - Since recalculateFinalMark( ) was private, code outside the class was not able to call this method, and therefore it can be safely removed.
- So we don't have to change any code outside the class!

Compare Versions

Student
– id : int – midterm : double – exam : double – forCredit : boolean – finalMark : double
+ getId( ) : int + setId(newID : int) + getMidterm( ) : double + setMidterm(newMark: double) + getExam( ) : double + setExam(newMark: double) + getForCredit( ) : boolean + setForCredit(newValue : boolean) + getFinalMark( ): double – recalculateFinalMark( )

Student
– id : int – midterm : double – exam : double – forCredit : boolean
+ getId( ) : int + setId(newID : int) + getMidterm( ) : double + setMidterm(newMark: double) + getExam( ) : double + setExam(newMark: double) + getForCredit( ) : boolean + setForCredit(newValue : boolean) + getFinalMark( ): double

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?

Initialization of Objects

- When we create a new Student, we used "new"

  aStudent = new Student( );

  The method Student() is called the default constructor of the class

- A constructor is a method that is executed when an object is created;

- A constructor has the same name as its class;
- A constructor can have arity 0, 1, 2, etc.
- Can only be called once, when the object is created, in the context \new . . . ";
- A constructor has no return type.

- Since the constructor is called when the object is first created, a constructor generally serves to initialize the instance variables.

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 (when should it be private);
  - 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.

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);
```

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)

```
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.

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** contains the value **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;
   }
}
```