"Politics is the skilled use of blunt objects."
-- L.B. Pearson

Section 10: Introduction to Objects

Objectives:

- Records
- Classes and Objects
- Information Hiding
- Accessors and Modifiers

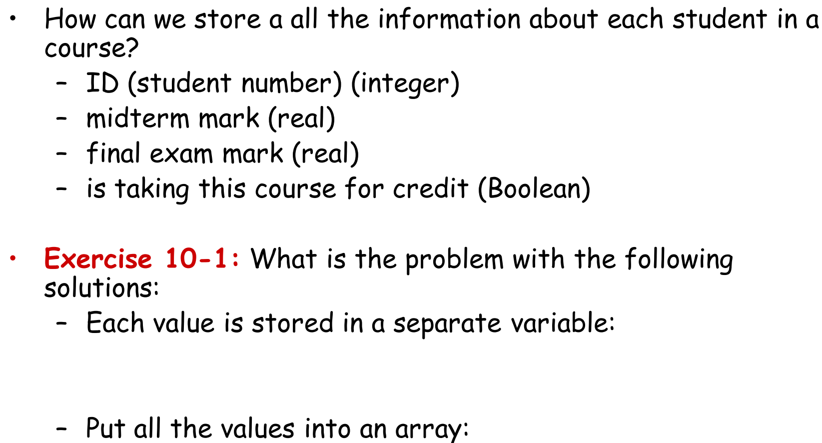
305

Historical note …

- Barbara Liskov was the first women to have obtained her Ph.D. in computer science in US (in 1968, from Stanford University).
- She was at the origin of CLU, the first language that supported abstract data types (1975), that influenced many object-oriented languages, including Java.
- In 1993, she and Janette Wing have developed a specific definition of sub-types, the Liskov principle of substitution, used in object-oriented programming.

306

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)

- **Exercise 10-1:** What is the problem with the following solutions:
  - Each value is stored in a separate variable:
  - Put all the values into an array:

?

Records

- Like an array, a "record" allows several values of different type to be stored in one variable.
  - Another view is that a record is a group of variables of different type
- Records differ from arrays in 2 ways:
  - The values/variables (called fields) in a record can be of different types.
  - Each field in a record has a NAME. A value is accessed by specifying the field name (not a subscript).
- Example (a single record with 4 fields):

field name	field value
id	1234567
midterm	60.0
exam	80.0
forCredit	TRUE

Using Records

- Suppose the preceding record was stored in a variable named r.
- To access the midterm mark:

  r.midterm

- This refers to one field inside record r. A field can be used anywhere a variable of that type is allowed, e.g.,

  t ← r.midterm + r.exam

  r.forCredit ← false

- The whole record can be used in an assignment statement or passed as a parameter:

  x ← r

(not in Java – this occurs in other programming languages like C)

Defining a Record Type

- When we discussed primitive types, we looked at:
  - What values does the type allow?
  - What operations one can do with values of that type?

- A record is a "user-defined" type that is built using types we have already:
  - Primitive types
  - Other user-defined types.

- Creating a record also has the two elements of primitive types:
  - What are the components of a record value?
  - What operations can we do with the record?

Records and Classes/Objects

- Some languages allow you to create records, without the capability to define operations on those records.
  - Examples:
    - The language Pascal has "records"
    - The language C has "structures"

- Some languages allow you to also define operations on user-defined data types.
  - The record types are then usually referred to as "classes", and specific records are called "objects"
  - Examples:
    - "Classes" in the language C++ and Java

Classes and Objects

- An object can be considered to be like a record, in that there is a set of "attributes" – named data values (variables) stored in the object.

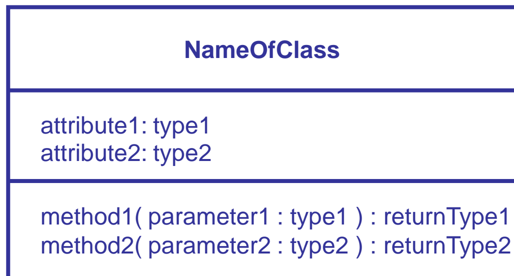
- Each object is created from a class. An object is referenced from a reference variable (like an array).

- 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).

- A method is called on an object using the **.** operator, in a similar manner to accessing a record field

  result ← anObject.aMethod(aParameter)

Class Diagrams



"Attributes" are like the field variables in a record

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

Translation to Java

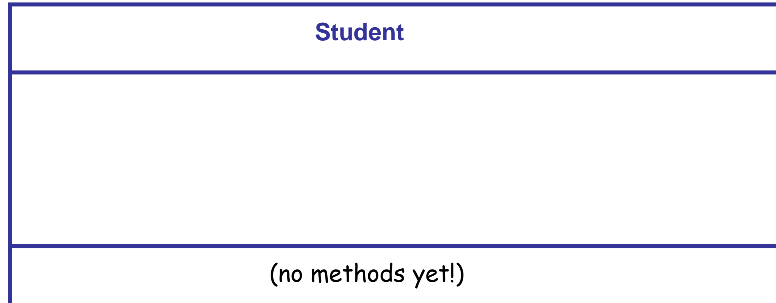
```
public class <Class Name>
{
   // Declaration of Variables
   // public <type> <name>;

   // Methods
}
```

- The class is a "template" for how to construct objects.
  - Objects must be created using the `new` statement
  - Objects are referenced with a reference variable

```
// Declaring the reference variable
<Class Name> refVar;
// creating the object
aStudent = new <Class Name>( );
```

Exercise 10-2: 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.*



Exercise 10-2: Translation to Java

```
public class Student
{



   // methods
}
// Declare aStudent reference Variable

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

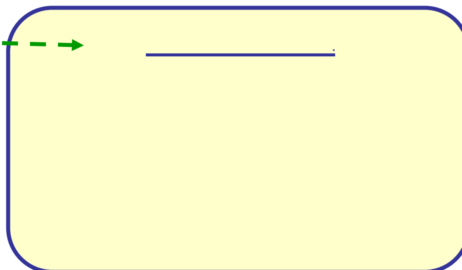
Exercise 10-3: Object usage in Java

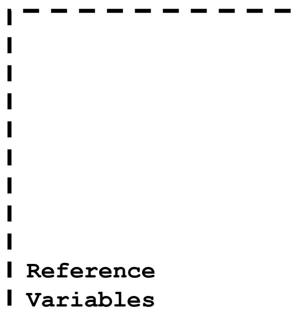
```
Student aStudent;           // declare reference variable
aStudent = new Student();   // create new object
aStudent.id = 1234567;
aStudent.midterm = 60.0;
aStudent.exam = 80.0;
aStudent.forCredit = true;

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


Exercise 10-3: Object usage in Java

format:
<class name>

(the underlining shows that this is an instance diagram)





Information hiding

- 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)

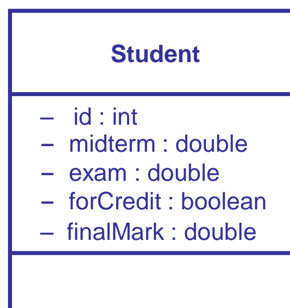
  is always true for consistency.

- It would be useful to prevent anyone else from setting the value of finalMark arbitrarily.
  - Instead, if the final mark is to change, it should be done by changing the value of either midterm or exam.

- Restricting access to data is called "information hiding".

319

Private fields in a class



- 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).

320

Information Hiding

- The field names and types represent an *implementation* of a class.
  - To ensure relative 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.

321

Second version of the Student Class

- This time, use encapsulation.

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

322

How do we use the second version?

- If we try the following:

```
Student aStudent = new Student();
aStudent.id = 1234567 ; // error!
```

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

- However, 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.

323

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).

324

Accessors and Modifiers

- Examples for the forCredit field in the class:

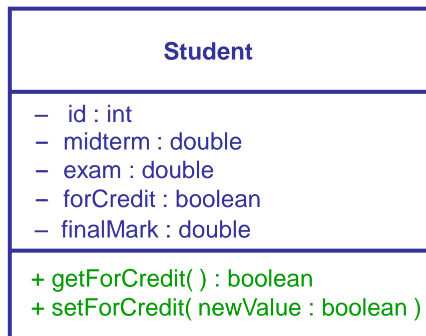
  + getForCredit( ) : boolean

    - 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.

  + setForCredit(newValue : boolean)

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

Class diagram with accessors and modifiers

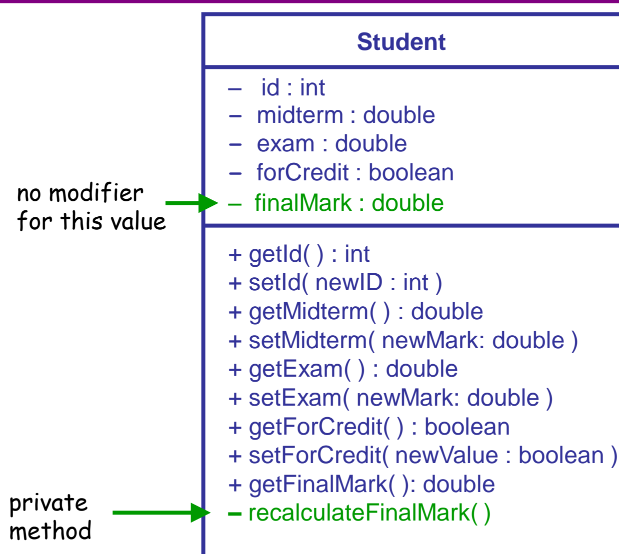


Back to Information Hiding

- To implement our strategy of hiding the finalMark field, we can do the following:
  - 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.

327

Student class with Information Hiding



328

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
```


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 );
```

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
}
```

331

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.

332

Methods Operate on Object Data

- Adding a method to a class such as Student provides operations on the data in the objects created with the class (user defined types).
- Note that we have called the method getForCredit without parameters in two different cases
  - Each different case calls to the methods are associated to different objects and thus we get different results.
- An analogie:
  - With integers: 3 + 5 and 4 + 3; the same operation (+) is invoked, but with different values which gives different results
  - With Students: the same operation (getForCredit), but on different objects ⇒ different results

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;
   }
}
```

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

`this`, again

- 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:
    - `exam` can be used instead of `this.exam`.
    - `recalculateFinalMark()` can be used instead of `this.recalculateFinalMark()`.

- There are 2 occasions when we really do need `this`:
  1. An object wants to pass itself as a parameter to a method of another class.
  2. An object wants to return a reference to itself as the result of a method.

338