Implementing classes

- From last time: Scanner example: **`TestReadLine.java`**
- Example: **`Student`** class
- instance variables
- method definitions
- scope and lifetime of variables
- public vs. private
- constructors

Announcements

- Staff office hours are pinned to the top of the piazza page
- PA1 will be available later this week. Due Wed. 2/10.
- Students who missed first class, who are not officially enrolled, or have no previous experience need to see me after class or in office hours today.

From last time…

How do you write code that combines word-by-word reading (e.g., with **`nextInt()`** or **`next()`**) and line-by-line reading (i.e., with **`nextLine()`**)?

- Specific example: write code that when run does…

```
Enter your age: 32
Enter your whole name: Joseph P. Blow
```

See **`TestReadLine.java`**

POLL: Review of object references / String

```
String greeting = "Hello";
String greeting2 = greeting;
```

How to take the poll:

During lecture: `pollev.com/cbono`

Asynchronous participation: Link to String poll

class example: `Student`

- **`Student`** class stores information about a student in a course.
- Keeps track of name and total score.
- Allows client to add scores to total.

Student class interface

```
Student stud = new Student("Joe");
      // has name "Joe" and total score of 0

Student blank = new Student();
      // has empty name and total score of 0

String name = stud.getName();

int total = stud.getTotalScore();

stud.addQuiz(score);
      // adds quiz score to the total
      //for this student
```

Complete code in **`Student.java`**

Instance variables

- a.k.a., fields, data members

```
public class Student {
    private String theName;
    private int totalScore;
    . . .
}
```

Using instance vars in a method

```
public class Student {
    private String theName;
    private int totalScore;
    . . .
    public void addQuiz(int score) {
        totalScore = totalScore + score;
    }
    . . .
}
```

class def

```
Student stud = new Student("Joe");
Student stud2 = new Student("Mary");
stud.addQuiz(7);
stud2.addQuiz(9);
```

client code

Scope of variables

```
public class Student {
    private String theName;
    private int totalScore;
   . . .
    public void addQuiz(int score) {
        totalScore = totalScore + score;
    }
    public int getTotalScore() {
        return totalScore;
    } . . .
}
```

class def

```
Student stud = new Student("Joe");
stud.addQuiz(7);
stud.addQuiz(10);
int tot = stud.getTotalScore();
int tot2 = totalScore;      // error (undefined)
int score2 = score;         // error (undefined)
```

client code

Scope: **local vars**

```
public class Student {
   . . . // Note: addBonus not part of actual class

   public void addBonus(int percent) {
      int bonus = Math.round(
                        totalScore*(percent/100.0));
      totalScore = totalScore + bonus;
    }
     . . .
}
```

```
Student stud = new Student("Joe");
stud.addQuiz(7);
stud.addQuiz(10);
stud.addBonus(10);
```

client code

Poll: New version of addBonus

```
public class Student {
    . . . // Note: addBonus not part of actual class

    public void addBonus(int percent) {
       int bonus = Math.round(
                        totalScore*(percent/100));
       totalScore = totalScore + bonus;
    }
      . . .
}
```

```
Student stud = new Student("Joe");
stud.addQuiz(7);
stud.addQuiz(10);
stud.addBonus(10);
```

client code

Asynchronous participation: Link to addBonus poll

Visibility of private instance vars

```
public class Student {
    private String theName;
    private int totalScore;
    . . .
    public void addQuiz(int score) {
        totalScore = totalScore + score;
    } . . .
}
```

class def

```
Student stud = new Student("Joe");
stud.addQuiz(7);
stud.addQuiz(10);
int tot = stud.totalScore; // error (pvt)
totalScore = 10; // error (undefined)
```

client code

Why do we make instance variables private?

Constructors

- Purpose: initialize all fields in the object
- Other methods can assume object is in a valid state.
- call: `new Student("Joe");`
  - internally initializes all the fields of this Student object.

Constructor example

```
public class Student {
    private String theName;
    private int totalScore;
   . . .
    public Student(String name) {
        theName = name;
        totalScore = 0;
    }
    . . .
}
```

class def

```
Student stud = new Student("Joe");
```

client code

Default initialization

- Without any constructors
  or before constructor executed:
  instance vars are assigned "default values"
  - `0` for numbers; **`null`** for obj. refs
  - recall: *local* variables are not automatically initialized
- better to initialize them explicitly

What is **`null`**?

- Default init. of object ref. *instance var.* is **`null`**
- **`null`** example: (shown with local var)

```
int len;  // Example with local variables
String s;                 // locals are not init'd
len = s.length();         // crashes
s = null;
len = s.length();         // crashes
if (s != null) {
  len = s.length();       // safe call
}
```

- **`null`** ≠ empty string

```
s = "";                   // empty string
len = s.length();         // 0
```

Default constructor

- The constructor with no parameters
- Do we need to write any code in the body of our **`Student`** default constructor?
- No-code version:

```
// creates student with empty name and
// total score of 0
public Student() {  }
```

Shadowing

Suppose instance variable identifier was the same as a parameter (or local var) identifier:

```
public class Student {   // this code doesn't work
    private String name;
    private int totalScore;
     . . .
    public Student(String name) {
        name = name;
        totalScore = 0;
    }
    . . .
}
```

Asynchronous participation: Link to Shadowing poll

One solution for Shadowing

Correct code where field identifier is the same as the parameter identifier:

```
public class Student {
    private String name;
    private int totalScore;
    . . .
    public Student(String name) {
        this.name = name;
        totalScore = 0;
    }
    . . .
}
```

Test programs

- **`Student`** class is not a complete program
- Could use in some larger app, or…
- first create a class to test the **`Student`** class:
  - **`StudentTester`**, will contain **`main`** method.
  - each class in it's own file:
    this one in **`StudentTester.java`**