CREATING CLASSES

ADDING A CONSTRUCTOR EXERCISE

- Create another constructor in the Student class.
  - That takes both the name and an array of grades.
  - To create an array of doubles for the grade array use:
    ```java
    double [] grades = {90.3, 85.2, 89.7, 95.3};
    ```
  - Use the debugger to check what happens during execution of this constructor.
CALCULATING THE GRADE AVERAGE

• Now that a student has an array of grades, one of the things we probably would like is to
  – Show the average when you print information about the student
• To calculate an average:
  – Sum the grades
  – Divide by the number of grades
    • The length of the array
• We need to be careful of
  – A null gradeArray
  – A 0 length gradeArray

CREATE A METHOD EXERCISE

• Create a method (getAverage) that will calculate and return the average of the grades in the grade array
  – It will return 0 if the grade array is null.
  – It will return 0 if the grade array length is 0.
  – Otherwise it will return
    (the sum of the grades) / (the number of grades).
• Add to the toString method a call to this method.
• Use the debugger to check that this is working correctly:
  – Stop in the getAverage method.

ACCESSING FIELDS FROM OTHER CLASSES

• Fields are usually declared to be private
  – So that code in other classes can’t directly access and change the data.

ACCESSING FIELDS FROM OTHER CLASSES

• Try this in the interactions pane
  > Student student1 = new Student("Sue Clark");
  > System.out.println(student1.name);
• You will get an exception
  – Short for exceptional event – i.e. an error
• In general outside classes cannot use
  object.field
  to access a private field value.
ACCESSORS AND MODIFIERS

- Rather than allowing direct access to the data fields inside a class, most classes are designed with methods to get or change the class fields' values.

ACCESSORS AND MODIFIERS

- The methods to get the value of a field are called **accessors**.
- The methods to change the value of a field are called **modifiers**.

ACCESSORS AND MODIFIERS

- Two reasons to use accessors and modifiers is that
  - the methods can be used to control access to the information, and
  - the modifiers can check that new values have valid values
    - *e.g.* a test score is between 0 and 100.
    - A RGB color is between 0 and 255.

ACCESSORS AND MODIFIERS

- Accessors
  - public methods that return data.
  - They are a safe way to access the data fields in a class.
    - In such a way as to protect the data for this object, by not allowing direct access to the field.
    - Their names usually begin with "get..."
  - Syntax:
    ```java
define fieldType getFieldName()
```
  - Example:
    ```java
    public String getName() {
        return this.name;
    }
    ```
ACCESSORS AND MODIFIERS

• Modifiers
  — public methods that modify the data fields in a class.
  — In such a way as to protect the data for this object.
    • Again, by not allowing direct access to the field data.
    • They usually begin with “set...”.
  — Syntax:
    ```java
    public returnType setFieldName(type name);
    ```
  — Example:
    ```java
    public void setName(String name) {
        this.name = name;
    }
    ```

CREATING STUDENT ACCESSORS

• Add a method to get the name:
  ```java
  public String getName() {
    return this.name;
  }
  ```

• What about a method to get the array of grades?
  — It’s not safe to return an array in the same way that it is to return a single value.
  — If someone gets the array they can directly change the grades in the array -
    • Since an array variable is an object, and points to the actual array.
  — It is safer to return an individual grade at an index value (e.g. the 0th grade, the 4th grade).
    — Then other classes can’t directly change the grade
  ```java
  public double getGrade(int index) {
    return this.gradeArray[index];
  }
  ```
We need some public methods
- That let other classes ask for the grade to change or the name to change.
- Our class is responsible for making sure this happens in ways that
  - Keeps data valid.
  - Avoids errors.

Setting a grade
- The grade must be ≥ 0
- The gradeArray must not be null
- The index must be < the length of the array

Setting a name
- You only want to allow a new name if it is null.
  - That is, no re-naming.

NAME MODIFIER

- Setting the name only if currently null

```java
public boolean setName(String name)
{
    if(this.name == null) {
        this.name = name;
        return true;
    } else return false;
}
```

GRADE MODIFIER

```java
public boolean setGrade(int index, double grade)
{
    if(grade < 0 || // Illegal value
        this.grades == null || // No array
        this.grades.length <= index || // Index too big
        index < 0) {
        return false;
    } else {
        this.grades[index] = grade;
        return true;
    }
}
```
ADD A FIELD EXERCISE

• Add a picture field to student that will hold a Picture object of a student.
• Add an accessor to get the value of this field.
• Add a modifier to set the value of this field.
• Add a method (i.e. show() ) that will show the picture.
  — If it isn’t null.

MAIN METHOD

• Add a main method to Student
  — Do in it what you have been doing in the interactions pane.

```java
public static void main(String[] args) {
    Student student1 = new Student();
    System.out.println(student1);
    Student student2 = new Student("Sue Clark");
    System.out.println(student2);
}
```

ICLICLICKER QUESTION

A. Object
B. Class
C. Java
D. Picture
E. None of the Above

All classes are descended from what class?