CSE 8A Lecture 16

- Reading for next class: None (in-term exam 4)
- Today’s goals:
  - Learn how to design and implement your own classes
- PSA 8 due tonight
- Lab Tomorrow!

- VERY IMPORTANT
  - LAST CHANCE to get RE-GRADE on previous PSAs
    (CHECK PIAZZA for detailed instructions)
WIC@UCSD
Beginner's Programming Competition

Saturday, November 23
3pm to 7pm
CSE B230
Teams of 2
Java
Pizza and Prizes!

Sign Up Today At:
wic.ucsd.edu/competition.html
1. Under what circumstances can you call a no-argument constructor WITHOUT explicitly defining it in the class?

A. Never. You must always define a no-argument constructor if you want to call it.

B. Only if you have NOT defined constructors that take arguments 60%

C. Only if you HAVE also defined constructors that take arguments.

D. Always. It is never necessary to define a no-argument constructor 20%
2. According to the textbook what is the usual return type for a “setter” method?

A. There is no one “common” return type, it depends.

B. Boolean -- it returns whether the change of instance variable was successful or not.

C. The type of the instance variable being set
3. What is the purpose of a `toString()` method in any class you design?

A. It can only return the type of the Object

B. It returns a String object with the information, usually about the object

C. It is necessary to have one, just because it is a Java rule
4. Is it possible to overload a constructor the same way you overload a method?

A. Yes

B. No, you can not overload a constructor the same way you overload a method

C. No, it is impossible to overload a constructor
About the midterm 4

• It will again ask you to do some coding

• It will focus on recent material (sounds) but of course you’ll need to know loops, conditionals, etc.

• Practice the following in particular (you have exercises to practice from previous class and at the end of these slides):
  – Writing code from scratch (examples given on next slides—you can generate more)
  – Tracing code (use the questions from the last few classes and modify them slightly) (with arrays especially)
Options to raisePitch

• Create new Sound (V1 solution in last class)
  – V1) Of exact length needed for higher pitched sound
  – V2) Of same length as original with “silence” at end
public Sound raiseP() {
    Sound highP = new Sound(this);
    SoundSample[] original = this.getSamples();
    SoundSample[] higher = highP.getSamples();
    int newPlace = 0;
    for (int origI = 0; origI < original.length; origI+=2) {
        higher[newPlace].setValue( original[origI].getValue() );
        newPlace++;
    }
    for (; newPlace < higher.length; newPlace++) {
        higher[newPlace].setValue(0);
    }
    return highP;
}
How would the code below change the SoundSample array?

// In the sound class
public void mystery() {
    SoundSample[] original = this.getSamples();
    for (int index = 0; index < original.length; index++) {
        original[index].setValue(original[index/2].getValue());
    }
}

1) Solo: (60 sec)
2) Discuss: (2 min)
3) Group: (20 sec)

It causes an error

DRAW!!
Lowering the Pitch of the Sound

// In the sound class
public void lowerPitch() {
    SoundSample[] original = this.getSamples();
    for (int index = 0; index < original.length; index++) {
        original[index].setValue( original[index/2].getValue() );
    }
}

Problem: We are overwriting the values we need to use before we have used them!
Possible solutions?

<table>
<thead>
<tr>
<th>before</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>300</th>
<th>140</th>
<th>10</th>
<th>-40</th>
<th>-100</th>
<th>-250</th>
<th>-150</th>
</tr>
</thead>
<tbody>
<tr>
<td>after</td>
<td>100</td>
<td>100</td>
<td>150</td>
<td>150</td>
<td>200</td>
<td>200</td>
<td>300</td>
<td>300</td>
<td>140</td>
<td>140</td>
</tr>
</tbody>
</table>
// In the sound class
public void lowerPitch() {
    SoundSample[] original = this.getSamples();

    for (int index = 0; index < original.length; index++) {
        original[index].setValue(original[index/2].getValue());
    }
}
Another name for this method...

// In the sound class
public void stretchInPlace() {
    SoundSample[] original = this.getSamples();

    for (int index = original.length - 1; index >= 0; index--)
    {
        original[index].setValue(original[index/2].getValue());
    }
}

<table>
<thead>
<tr>
<th>before</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>300</th>
<th>140</th>
<th>10</th>
<th>-40</th>
<th>-100</th>
<th>-250</th>
<th>-150</th>
</tr>
</thead>
<tbody>
<tr>
<td>after</td>
<td>100</td>
<td>100</td>
<td>150</td>
<td>150</td>
<td>200</td>
<td>200</td>
<td>300</td>
<td>300</td>
<td>140</td>
<td>140</td>
</tr>
</tbody>
</table>
…that could apply to Pictures too!

**PRACTICE PROBLEM**: Complete the code below to stretch the calling object Picture both horizontally and vertically.

```java
// In the sound class
public void stretchInPlace() {
    SoundSample[] original = this.getSamples();
    for ( int index = original.length-1; index >= 0; index-- ) {
        original[index].setValue( original[index/2].getValue() );
    }
}

// In the **Picture** class
public void stretchInPlace() {

    for ( int x =
    {
        for ( int y =
        {
            Pixel source = this.getPixel()
            );

            Pixel target = this.getPixel()
            );

            target.setColor( source.getColor() );
        }
    }
}
```

Fill in!
Classes and objects

• **Classes and objects** are essential concepts in *object-oriented* programming languages like Java

• Classes and objects in programming languages are something like classes and objects in the real world

• Understanding the parallels can help you understand the concepts!
Objects

• Real world **objects** have:
  
  – **Properties**
    • For example, a car has a price, a color, an owner, a mileage rating, a location, a weight, an amount of gas in its tank…
  
  – **Behaviors**
    • For example, a car can be sold (change its owner), can move (change its location), can be filled (change the amount of gas in its tank)…

• **Software objects** have properties and behaviors too…
Classes

• Real world objects are instances of one or more classes
  – For example, any car object is an instance of the class “car”.
  – The class an object is an instance of determines what properties it has, and what behaviors it has
  – Note: it is possible for a class to exist, without any objects that are instances of it (example: the class of unicorns!). The class comes first.

• Software objects are instances of classes too…
Instance Variables and Methods

• Properties of an object are determined by its *instance variables*
  – What types are they? What values do they have?
  – (also known as ‘fields’ or ‘member variables’)

• Behaviors of an object are determined by its *instance methods*
  – What parameters do they take? What values do they return (if any)? What do they *do*?
  – *Constructors* are a special kind of method…
Visibility of Class Elements

- "private" means: visible only inside this class
- "public" means: everyone can "access" and call this element

Class design rule of thumb: **make all instance variables private** (and methods are typically public)

- So a private instance variable or instance method cannot be seen from outside the class
- Making an instance variable private prevents incorrectly setting its value by malicious or careless users of the class
Special methods: Constructor

• When an object is created, its instance variables need to be initialized

• This is the job of a constructor method

• Constructor methods in a class:
  – Always have the same name as the class
  – Never have a return type (not even void)
  – Can take parameters
  – Can be overloaded
  – Are only called when an object is instantiated
A **Point class**

**in a file named Point.java**

```
public class Point {
    private double x;
    private double y;
    
    public Point(double x_in, double y_in) {
        this.x = x_in;
        this.y = y_in;
    }

    public static void main(String[] args) {
        double d = 42.0;
        Point p;
        p = new Point(d, 42.0);
        
        Point q = new Point(p.x, 42);
    }
}
```

What are the *instance variables* in Point?

A. x, y  
B. x_in, y_in  
C. x, y, x_in, y_in  
D. p, q  
E. x, y, p, q
The Point of Java: objects and classes

A Point class

```java
public class Point {
    private double x;
    private double y;

    public Point(double x_in, double y_in) {
        this.x = x_in;
        this.y = y_in;
    }

    public static void main(String[] args) {
        double d = 42.0;
        Point p;
        p = new Point(d, 42.0);
        Point q = new Point(p.x, 42);
    }
}
```

How many arguments (parameters) does the Point constructor take?

A. 0
B. 1
C. 2
D. 0 or 2 (you can call it with either)
E. 0, 1 or 2
public class Point
{
    private double x;
    private double y;

    public Point(double x_in, double y_in)
    {
        this.x = x_in;
        this.y = y_in;
    }

    public static void main( String[] args )
    {
        double d = 42.0;
        Point p;
        p = new Point(d, 42.0);

        Point q = new Point(p.x, 42);
    }
}
double a = 42.0;
double b = a;

A. a 142.0
   b 142.0
B. a 142.0
   b 1
The `Point` class

```java
public class Point {
    private double x;
    private double y;

    public Point(double x_in, double y_in) {
        this.x = x_in;
        this.y = y_in;
    }

    public static void main(String[] args) {
        double d = 42.0;
        Point p;
        p = new Point(d, 42.0);

        Point q = new Point(p.x, 42);
        d = 65.0;
        p.x = 55.0;
        System.out.println(d + "", x + p.x + ", " + q.x);
    }
}
```

What does the code print?

A. 42.0, 42.0, 42.0  
B. 65.0, 55.0, 65.0  
C. 55.0, 55.0, 42.0  
D. 65.0, 55.0, 55.0  
E. 65.0, 55.0, 42.0
Class, Field, or Method?

• A class is a **type** of thing

• A field (instance variable) is a **property**, that might possibly have different values at different times

• A method is an **action** that can be performed

• So usually class and **field** names are **nouns**, **method** names are **verbs**…
Another Example: the Pixel class

Practice problem:

• Look at code in Pixel.java

• Identify: constructors, instance variables, instance methods
Another example of a class

• Class: \texttt{Species}
• Instance Variables:
  – \texttt{name}, a \texttt{String}
  – \texttt{population} on 7 continents, an array of 7 \texttt{ints}
  – \texttt{growthRate}, a \texttt{double}

• Constructor: one that takes no arguments, and:
  – initializes \texttt{name} to “No Name Yet”
  – initializes \texttt{population} to all 0
  – initializes \texttt{growthRate} to 33.3
How many errors are there in this code (and what are they)

A. 2
B. 3
C. 4
D. 5
E. >=6

```java
public class Species

    private String name;
    {

    public static void main(String[] args)
    {
        double[] population;
        double growthRate;
    }

    public Species()
    {
        String name = "No Name Yet";
        double[] population = {0,0,0,0,0,0,0,0};
        growthRate = 33.3;
    }

    }
```
public class Species
{
    private String name;

    public static void main(String[] args)
    {
        private int[] population;
        private double growthRate;

        Instance variables should be private

        public Species()
        {
            String name = "No Name Yet";
            int[] population = {0,0,0,0,0,0,0,0};
            growthRate = 33.3;
        }
    }
}

All instance variables and methods have to go inside the class {}

All instance variables have to go outside of method definitions

Instance variables should be private

Creating an array like this doesn’t work unless its with the declaration

Refer to instance variables in a constructor; do not redeclare them
public class Species
{

    /////////// fields ///////////
    private String name;
    private int[] population;
    private double growthRate;

    /////////// constructors ///////////
    public Species()
    {
        String name = "No Name Yet";
        population = {0,0,0,0,0,0,0};
        growthRate = 33.3;
    }

    /////////// methods ///////////
}
Getter and Setter methods

• Q: Instance variables correspond to properties of an object... if they are private and hidden inside, how can they interact with other objects?

• A: Define public instance methods which give controlled, safe access to the private instance variables

  – If the method can change an instance variable, it is a “mutator” or “modifier” or “setter” method
  – If it only returns the value of an instance variable, it is an “accessor” or “getter” method
Which of the following would you select for "getter" method signatures for Species class?

A. public void getName();
   public void getPopulation();
   public void getGrowthRate();

B. public String getName();
   public int[] getPopulation();
   public double getGrowthRate();

C. public void getName(String newName);
   public void getPopulation(int newPop);
   public void getGrowthRateRate(double newGrowthRate);

D. private String getName();
   private int[] getPopulation();
   private double getGrowthRate();

E. None of the Above
Which of the following would you select for "setter" method declarations for the `Species` class?

<table>
<thead>
<tr>
<th>Option</th>
<th>Method Declarations</th>
</tr>
</thead>
</table>
| A      | public void setName();
        | public void setPopulation();
        | public void setGrowthRate(); |
| B      | public String setName();
        | public int[] setPopulation();
        | public double setGrowthRate(); |
| C      | public void setName(String newName);
        | public void setPopulation(int[] newPop);
        | public void setGrowthRate(double newGrowthRate); |
| D      | public void setName(String newName);
        | public boolean setPopulation(int[] newPop);
        | public void setGrowthRate(double newGrowthRate); |
| E      | None of the Above    |
Return type for Getter and Setters

- A getter method should have a non-void return type.

- A setter can be designed in several ways:
  - **void**: just change the values of the instance variable(s), don’t return anything.
  - **boolean**: return true if the setting was successful and false if not (for example if setting would be ‘illegal’).
  - The type of the value that is being changed: return the previous value.
Overloading: Which are legal overloads?

A. 1
B. 2
C. 3
D. 1 and 3
E. 1 and 2

1) public Species()
   public Species(String newName);

2) public boolean setGrowthRate(double gr)
   public void setGrowthRate(double gr);

3) public void setPopulation(
       int northAmerica,
       int southAmerica,
       int europe,
       int asia,
       int africa,
       int australia,
       int antarctica)
   public void setPopulation(int[] a)
PRACTICE PROBLEM:
Write a method in the Sound class that takes two parameters, an int named skip and an int named offset, and, starting at offset, sets the value of every skip’th sample to 0 until the end. This method modifies the calling object and returns nothing. Be sure your code works when the offset is larger than the size of the array.

Here is an example of calling s.setSomeToZero( 4, 3 ) (where 4 is the skip and 3 is the offset):
s’s SoundSample array before

| 100 | 150 | 200 | 300 | 140 | 10 | -40 | -100 | -250 | -150 |

s’s SoundSample array after

| 100 | 150 | 200 | 0   | 140 | 10 | -40 | 0    | -250 | -150 |

Alternative easier version: Assume skip is always 4 and don’t pass it as a parameter (Do this version first if you’re having trouble with the version above). This alternative version is close to the difficulty of the problems on the exam. The original version is slightly harder than a problem on the exam.
PRACTICE PROBLEM:
Write a method in the Sound class that takes no arguments and reverses the calling object Sound \textit{in place} (i.e., without copying the sound sample array to a new array).
PRACTICE PROBLEM:
Write a method in the Sound class that takes two integer arguments, start and end, and returns a new Sound. The new sound is the same as the calling object Sound, but the interval between start and end (in the original Sound, up to but not including end) is double the frequency. The new Sound should be the same length as the calling object Sound, and the end will be filled with silence (0s). Here's an example of calling s.higherRange(2, 7):

s's SoundSample array

| 100 | 150 | 200 | 300 | 140 | 10 | -40 | -100 | -250 | -150 |

returned sound's SoundSample array

| 100 | 150 | 200 | 140 | -40 | -100 | -250 | -150 | 0    | 0    |

NOTE: slightly harder than an exam problem, but not by too much
TODO

• Finish PSA today!
• Lab and new PSA9 out tomorrow
• Prepare for midterm on thursday
Terminology Check

1. Declaration:  
   - A: Blue  
   - B: Blue  
   - C: Black  
   - D: Black  

2. Instantiation:  
   - A: Red  
   - B: Black  
   - C: Red  
   - D: Blue  

3. Initialization:  
   - A: Black  
   - B: Red  
   - C: Blue  
   - D: Red  

---

```java
foo = new double[5];
for (int i = 0; i < foo.length; i++)
    foo[i] = -11.5;
```

```
double [] foo;
```
public class Species{

    /////////// fields ///////////
    private String name;
    private int[] population;
    private double growthRate;

    //////////// constructors ///////////
    public Species(String name, int[] pop, double gr)
    {
        this.name = name;
        population = new int[pop.length];
        for (int i=0; i< this.population.length;i++)
            population[i] = pop[i];
        growthRate = gr;
    }

    ////////// methods ///////////
}
public Species(String newName, int[] newPop, double newGR) {
    name = newName;
    population = new int[newPop.length];
    for (int i=0; i< this.population.length;i++)
        population[i] = newPop[i];
    growthRate = newGR;
}
The Species class, continued

/////// methods /////////////////

public void setPopulation(int pop, int index) {
    population[index] = pop;
}

public int getPopulation(int index) {
    return population[index];
}

}
A redesign of the Species class

• This idea that the population array is just 7 entries, one per location is a bit “obscure”.
  – What are the names of the locations? Which entry is for North America? Which for Europe?

• Another, better approach: “parallel arrays”

• Declare and create two arrays of the same length
  – One for location names: String[] location;
  – One for population numbers: int[] population;
  – And write code so that for every index I, population[I] is the population in the location with name location[I]
public Species(String name, int[] pop, String[] location, double gr) {
    this.name = name;
    population = new int[pop.length];
    for (int i=0; i< population.length;i++)
        population[i] = pop[i];

    location = new String[location.length];
    for (int i=0; i < location.length; i++)
        location[i] = location[i];

    this.location = location;
    location = new String[location.length];
    for (int i=0; i < location.length; i++)
        location[i] = location[i];

    this.location = location;
}

<<INSERT CODE HERE>>
growthRate = gr;

A) location = location;
B) this.location = location;
C) location = new String[location.length];
for (int i=0; i < location.length; i++)
    location[i] = location[i];
D) location = new String[location.length];
for (int i=0; i < location.length; i++)
    this.location[i] = location[i];
E) None of the above
Write a constructor for the new Species class

```java
public Species(String name, int[] pop, String[] location, double gr) {
    this.name = name;
    population = new int[pop.length];
    for (int i=0; i< population.length;i++)
        population[i] = pop[i];

    this.location = new String[location.length];
    for (int i=0; i < location.length; i++)
        this.location[i] = location[i];

    growthRate = gr;
}
```

Still has some “bad software design issues…”
public Species(String name, int[] pop, String[] location, double gr)
{
    name = newName;
    growthRate = gr;
    if (pop.length != location.length)
    {
        System.out.println("Error constructing Species. " + "Population array and location array must be same length.");
        population = null; location = null;
        return;
    }
    population = new int[pop.length];
    this.location =
        new String[location.length];
    for (int i=0; i < location.length; i++)
    {
        this.location[i] = location[i];
        this.population[i] = pop[i];
    }
}
public boolean setPopulation(int pop, String loc) {
    if (pop < 0) return false;
    for (int i=0; i<loc.length; i++)
    {
        if (location[i].equals(loc))
        {
            population[i] = pop;
            return true;
        }
    }
    return false;
}