CSE 8A Lecture 17

• Reading for next class: review chapter 11

• Today’s goals:
  – Static methods
  – Getter/Setter
  – More practice with designing classes
  – Tracing code and creating memory models

How was the exam?
A. Horrible!  B. Bad  C. So-So  D. Not Too Bad  E. Not Bad at all
Unraveling the magic of **main**

"visibility" of this method

Doesn’t return anything

**public static void main(String[] args)**

Parameters (how are these passed in?)

Method name

Who “owns” the method...
Classes vs. Objects

In the code below, how many classes are there? How many objects? (And what does the memory model look like?)

```java
Picture p = new Picture( "fish.jpg" );
Picture p2 = new Picture( p );
Picture p3 = p2;
```

A. 1 class, 2 objects  
B. 1 class, 3 objects  
C. 2 classes, 2 objects  
D. 2 classes, 3 objects  
E. 3 classes, 2 objects
Classes vs. Objects

In the code below, how many classes are there? How many objects? (And what does the memory model look like?)

```java
Picture p = new Picture("fish.jpg");
Picture p2 = new Picture(p);
Picture p3 = p2;
```

The `Picture` class

The diagram shows the `Picture` class and its objects and variables. The variables `p`, `p2`, and `p3` point to the same object in memory, demonstrating aliasing. The `new` keyword is used to create new objects, and the class `Picture` is instantiated with the constructor `new Picture("fish.jpg")`. The memory model is illustrated with arrows pointing from the variables to the objects, showing the relationships between them.
public class Picture{
    ...
    public static Picture collage(Picture p1, Picture p2, Picture p3)
    public void filter1()
    }

static
Picture result = Picture.collage(p1, p2, p3);

(class - owned)

(nonstatic)

Picture p = new Picture("fish.jpg");
p.filter1();
Static vs. nonstatic methods

**Static** methods can be called directly on the **class** (can also be called on an object)
**Non-static** methods must be called on an object

```java
class Picture {
    public static Picture collage(Picture p1, Picture p2, Picture p3) {
        // Collage three pictures
    }
    public void filter1() {
        // Filter the picture
    }
}
```

**Class-owned**
- `public static Picture collage(Picture p1, Picture p2, Picture p3)`
- `public void filter1()`

**Object-owned**

```java
static Picture result = Picture.collage(p1, p2, p3);
```

```java
Picture p = new Picture("fish.jpg");
p.filter1();
```

- `result` is a `Picture` object returned by `collage` method.
- `p` is an `Object` instance of `Picture` class.
Unraveling the magic of `main`

`public static void main(String[] args)`

- "visibility" of this method
- Doesn't return anything
- Method name
- Who "owns" the method...

`Lab8.main( )`

Why does `main` have to be `static`? (Discuss with group)
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
public class Species
{

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

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

///////// methods ///////////
}
Which of 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 getGrowthRate(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?

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
1. Declaration: 
   
2. Instantiation: 
   
3. Initialization: 
   
```
for (int i = 0; i < foo.length; i++)
{ foo[i] = -11.5; }
```

```
double [] foo;
```
Overloading: Which are legal overloads?

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

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

2)
```java
public boolean setGrowthRate(double gr)
public void setGrowthRate(double gr)
```

3)
```java
public void setPopulation(
    int northAmerica,
    int southAmerica,
    int europe,
    int asia,
    int africa,
    int australia,
    int antarctica)
public void setPopulation(int[] a)
```
The Species class, another constructor

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;
}
// ...In main
int[] pops = {20, 30, 1, 4, 2, 6, 11};
Species s = new Species( "Oompa", pops, 1.2 );
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];

    growthRate = gr;
}

A. location = location;
B. location = new String[location.length];
   for (int i=0; i < location.length; i++)
       location[i] = location[i];
C. this.location = location;
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;
}
• Reading for next class: Review chapter 11
• Get started with PSA 9 asap!