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
- Parameters (how are these passed in?)

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

- 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
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;
```
public class Picture{
    ...
    public static Picture collage(Picture p1, Picture p2, Picture p3)
    public void filter1()
}

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

(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
public class Picture{
    ...  
    public static Picture collage(Picture p1, Picture p2, Picture p3)
    public void filter1()
}
```

**Static** (class-owned)
```
static
Picture result = Picture.collage(p1, p2, p3);
```

**Non-static** (object-owned)
```
Picture p = new Picture("fish.jpg");
p.filter1();
```
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?)
- Who "owns" the method…
- 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 the following would you select for "getter" method signatures for the Species class?

A

```java
public void getName();
public void getPopulation();
public void getGrowthRate();
```

B

```java
public String getName();
public int[] getPopulation();
public double getGrowthRate();
```

C

```java
public void getName(String newName);
public void getPopulation(int newPop);
public void getGrowthRate(double newGrowthRate);
```

D

```java
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 Species class?

A

```java
public void setName();
public void setPopulation();
public void setGrowthRate();
```

B

```java
public String setName();
public int[] setPopulation();
public double setGrowthRate();
```

C

```java
public void setName(String newName);
public void setPopulation(int[] newPop);
public void setGrowthRateRate(double newGrowthRate);
```

D

```java
public void setName(String newName);
public boolean setPopulation(int[] newPop);
public void setGrowthRateRate(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
Terminology Check

1. Declaration: Blue  Blue  Black  Black
2. Instantiation: Red   Black  Red   Blue
3. Initialization: Black  Red   Blue  Red

E. None of the Above

`foo = new double[5];`

```
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

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

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

```
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

```java
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]
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];

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

```java
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!