Collections as Fields

Still Aggregation and Composition

Motivation

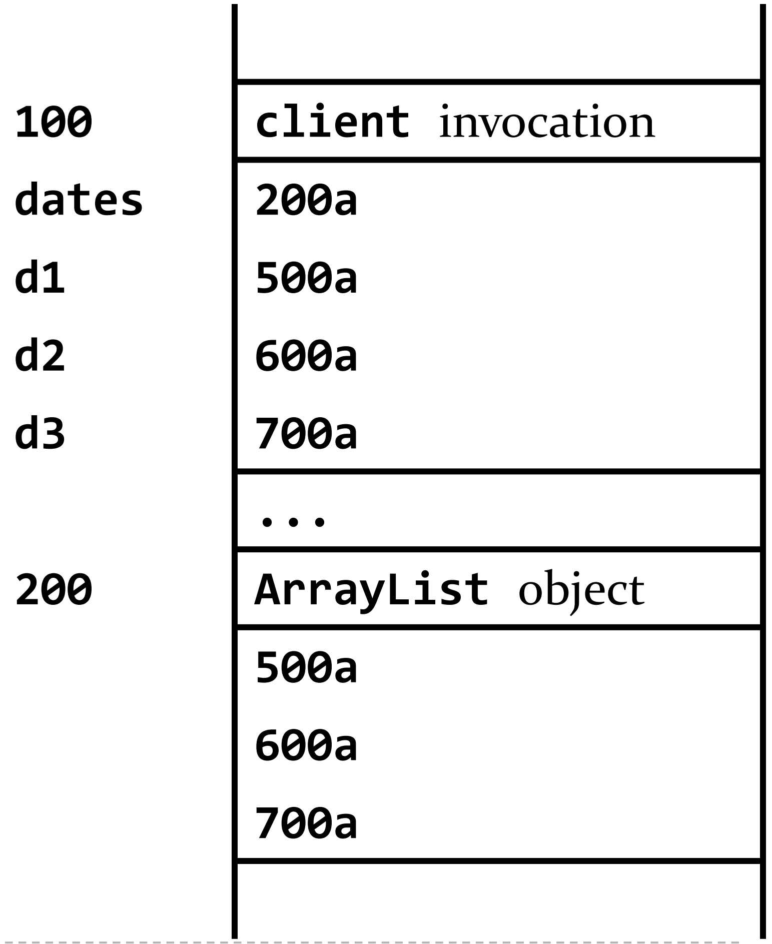
- often you will want to implement a class that has-a collection as a field
  - a university has-a collection of faculties and each faculty has-a collection of schools and departments
  - a receipt has-a collection of items
  - a contact list has-a collection of contacts
  - from the notes, a student has-a collection of GPAs and has-a collection of courses
  - a polygonal model has-a collection of triangles*

*polygons, actually, but triangles are easier to work with

What Does a Collection Hold?

- a collection holds references to instances
  - it does not hold the instances

```
ArrayList<Date> dates =
        new ArrayList<Date>();

Date d1 = new Date();
Date d2 = new Date();
Date d3 = new Date();

dates.add(d1);
dates.add(d2);
dates.add(d3);
```



Test Your Knowledge

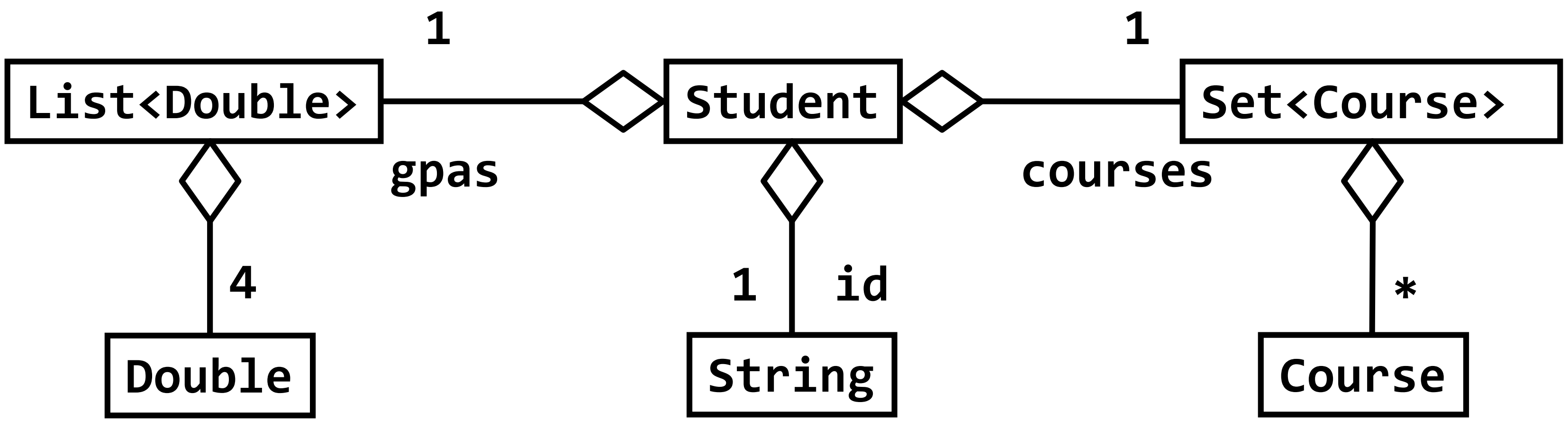
1. What does the following print?

```
ArrayList<Point> pts = new ArrayList<Point>();
Point p = new Point(0., 0., 0.);
pts.add(p);
p.setX( 10.0 );
System.out.println(p);
System.out.println(pts.get(0));
```

2. Is an **`ArrayList<X>`** an aggregation of **`X`** or a composition of **`X`**?

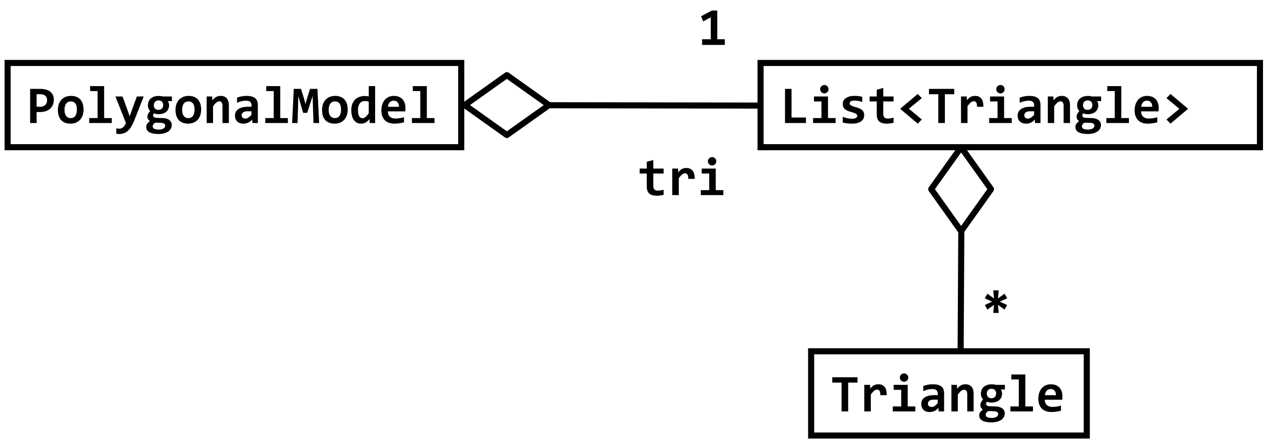
Student Class (from notes)

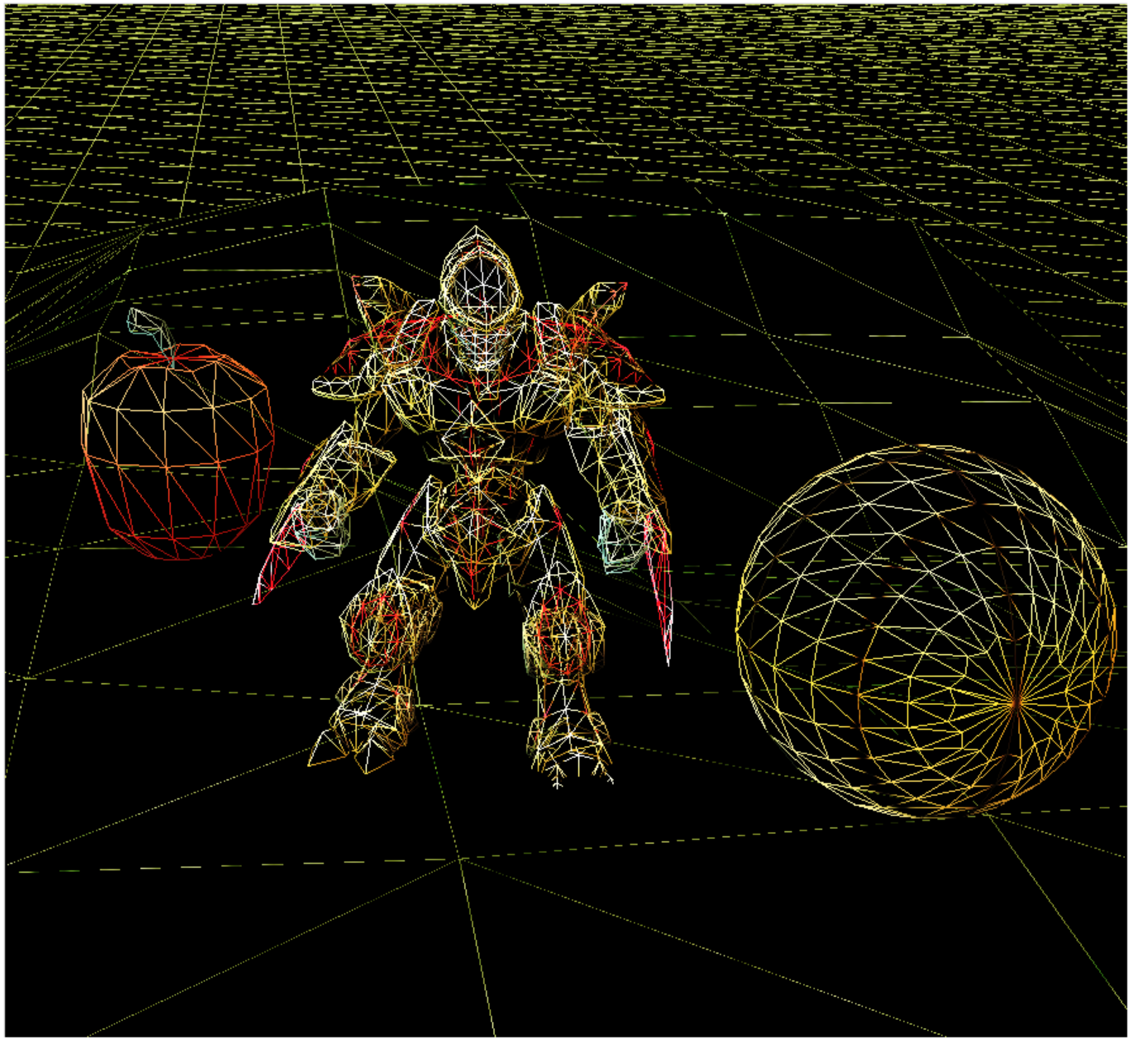
- a Student has-a string id
- a Student has-a collection of yearly GPAs
- a Student has-a collection of courses



PolygonalModel Class

- a polygonal model has-a **`List`** of **`Triangle`**s
  - aggregation





PolygonalModel

```
class PolygonalModel {

  private List<Triangle> tri;

  public PolygonalModel() {
      this.tri = new ArrayList<Triangle>();
  }


}
```

PolygonalModel

```
public void clear() {
    // removes all Triangles
    this.tri.clear();
}

public int size() {
    // returns the number of Triangles
    return this.tri.size();
}
```

Collections as Fields

- when using a collection as an attribute of a class **X** you need to decide on ownership issues
  - does **X** own or share its collection?
  - if **X** owns the collection, does **X** own the objects held in the collection?

X Shares its Collection with other **X**s

- if **X** shares its collection with other **X** instances, then the copy constructor does not need to create a new collection
  - the copy constructor can simply assign its collection
  - [notes 5.3.3] refer to this as aliasing

PolygonalModel Copy Constructor 1

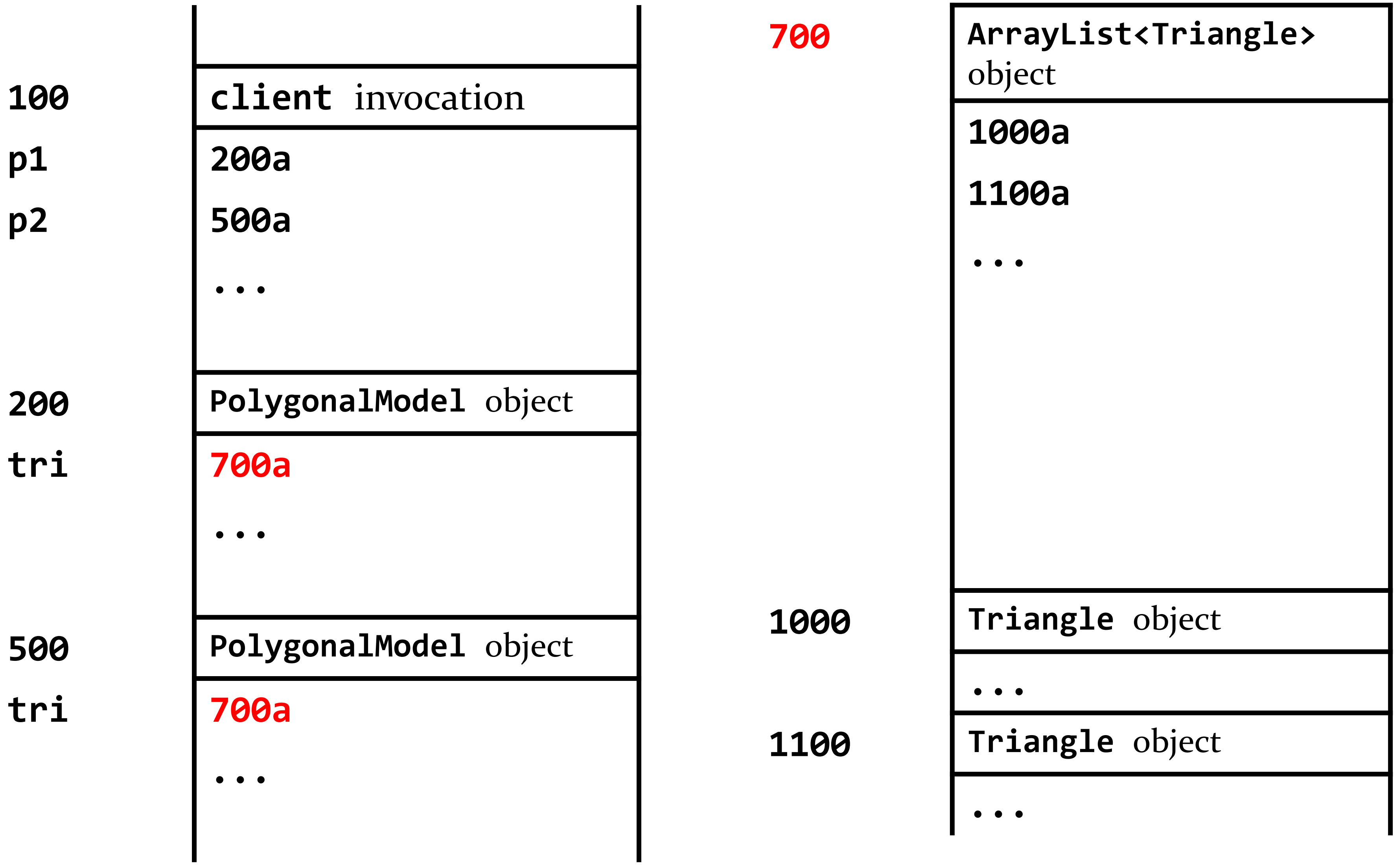
```
public PolygonalModel(PolygonalModel other) {
    // implements aliasing (sharing) with other
    //   PolygonalModel instances
    this.tri = other.tri;
}
```

alias: no new `List` created

```
public List<Triangle> getTriangles() {
    return this.tri;
}
```

alias: no new `List` created

```
PolygonalModel p2 = new PolygonalModel(p1);
```



Test Your Knowledge

1. Suppose that the **`PolygonalModel`** copy constructor makes an alias of the list of triangles.

   Suppose you have a **`PolygonalModel p1`** that has 100 **`Triangle`**s. What does the following code print?

```
PolygonalModel p2 = new PolygonalModel(p1);
p2.clear();
System.out.println( p2.size() );
System.out.println( p1.size() );
```

X Owns its Collection: Shallow Copy

- if **X** owns its collection but not the objects in the collection then the copy constructor can perform a shallow copy of the collection
- a shallow copy of a collection means
  - **X** creates a new collection
  - the references in the collection are aliases for references in the other collection

X Owns its Collection: Shallow Copy

- the hard way to perform a shallow copy of a list named **`dates`**

shallow copy: new **`List`** created but elements are all aliases

```
ArrayList<Date> sCopy = new ArrayList<Date>();
for(Date d : dates) {
    sCopy.add(d);
}
```

`add` adds an alias of **`d`** to **`sCopy`**

X Owns its Collection: Shallow Copy

- the easy way to perform a shallow copy of a list named **`dates`**

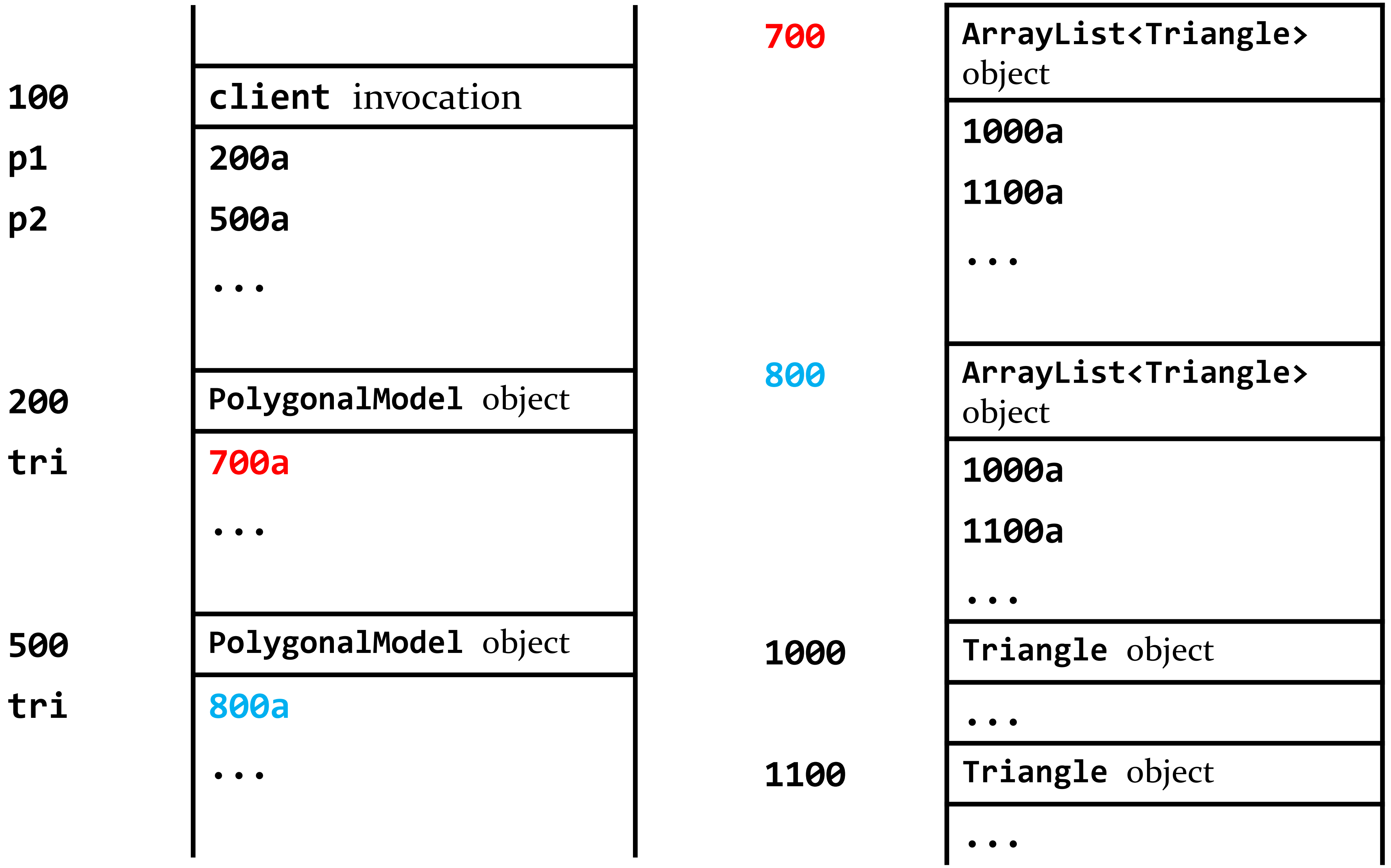
```
ArrayList<Date> sCopy = new ArrayList<Date>(dates);
```

PolygonalModel Copy Constructor 2

```
public PolygonalModel(PolygonalModel other) {
    // implements shallow copying
    this.tri = new ArrayList<Triangle>(other.tri);
}
```

shallow copy: new **`List`** created, but no new **`Triangle`** objects created

```
PolygonalModel p2 = new PolygonalModel(p1);
```



Test Your Knowledge

2. Suppose that the **PolygonalModel** copy constructor makes a shallow copy of the list of triangles.

   Suppose you have a **PolygonalModel p1** that has 100 **Triangle**s. What does the following code print?

```
PolygonalModel p2 = new PolygonalModel(p1);
p2.clear();
System.out.println( p2.size() );
System.out.println( p1.size() );
```

Test Your Knowledge

3. Suppose that the **PolygonalModel** copy constructor makes a shallow copy of the list of triangles.

   Suppose you have a **PolygonalModel p1** that has 100 **Triangle**s. What does the following code print?

```
PolygonalModel p2 = new PolygonalModel(p1);
Triangle t1 = p1.getTriangles().get(0);
Triangle t2 = p2.getTriangles().get(0);
System.out.println(t1 == t2);
```

X Owns its Collection: Deep Copy

- if **X** owns its collection and the objects in the collection then the copy constructor must perform a deep copy of the collection
- a deep copy of a collection means
  - **X** creates a new collection
  - the references in the collection are references to new objects (that are copies of the objects in other collection)

X Owns its Collection: Deep Copy

- how to perform a deep copy of a list named **`dates`**

```
ArrayList<Date> dCopy = new ArrayList<Date>();
for(Date d : dates) {
    dCopy.add(new Date(d.getTime());
}
```

deep copy: new **`List`** created and new elements created

new **`Date`** created that is a copy of **`d`**

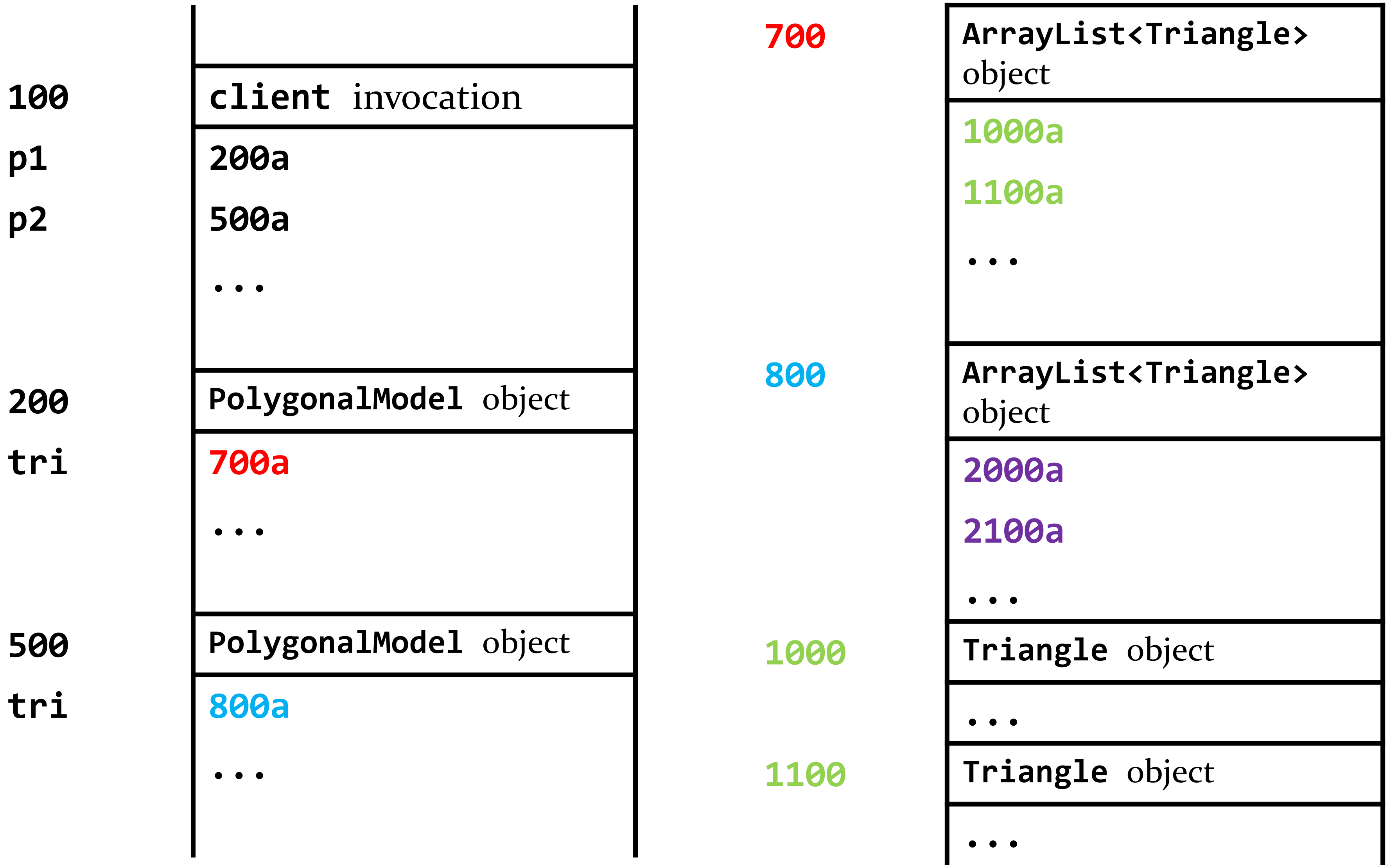
PolygonalModel Copy Constructor 3

```
public PolygonalModel(PolygonalModel other) {


    // implements deep copying
    this.tri = new ArrayList<Triangle>();
    for (Triangle t : other.getTriangles()) {
        this.tri.add(new Triangle(t));
    }
}
```

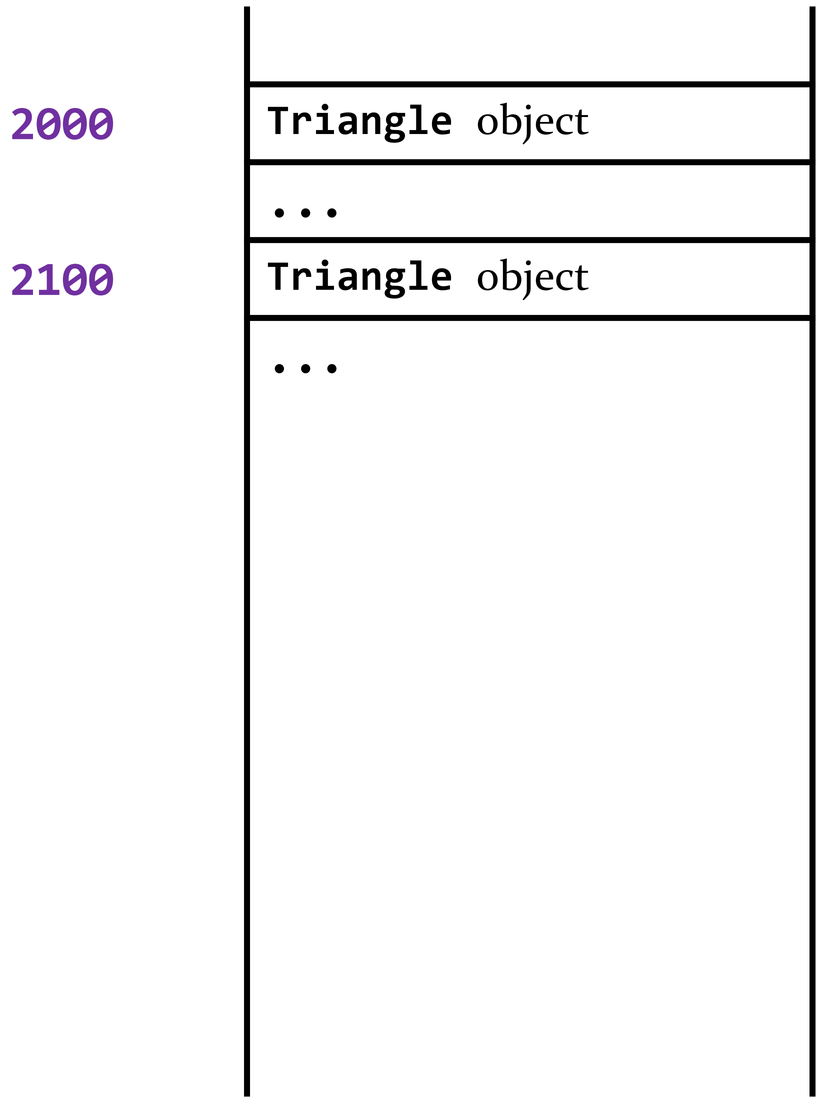
deep copy: new **`List`** created, and new **`Triangle`** objects created

new **`Triangle`** created that is a copy of **`t`**

```
PolygonalModel p2 = new PolygonalModel(p1);
```



continued on next slide


2000
`Triangle` object
...
2100
`Triangle` object
...

Test Your Knowledge

4. Suppose that the **`PolygonalModel`** copy constructor makes a deep copy of the list of triangles.

   Suppose you have a **`PolygonalModel p1`** that has 100 **`Triangle`**s. What does the following code print?

```
PolygonalModel p2 = new PolygonalModel(p1);
p2.clear();
System.out.println( p2.size() );
System.out.println( p1.size() );
```

Test Your Knowledge

5. Suppose that the **`PolygonalModel`** copy constructor makes a deep copy of the list of triangles.

   Suppose you have a **`PolygonalModel p1`** that has 100 **`Triangle`**s. What does the following code print?

```
PolygonalModel p2 = new PolygonalModel(p1);
Triangle t1 = p1.getTriangles().get(0);
Triangle t2 = p2.getTriangles().get(0);
System.out.println(t1 == t2);
System.out.println(t1.equals(t2));
```

Arrays

Arrays

- in Java an array is a container object that holds a fixed number of values of a single type
- the length of an array is established when the array is created

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Arrays

- to declare an array you use the element type followed by an empty pair of square brackets

```
double[] collection;
// collection is an array of double values


collection = new double[10];
// collection is an array of 10 double values
```

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Arrays

- to create an array you use the new operator followed by the element type followed by the length of the array in square brackets

```
double[] collection;
// collection is an array of double values


collection = new double[10];
// collection is an array of 10 double values
```

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Arrays

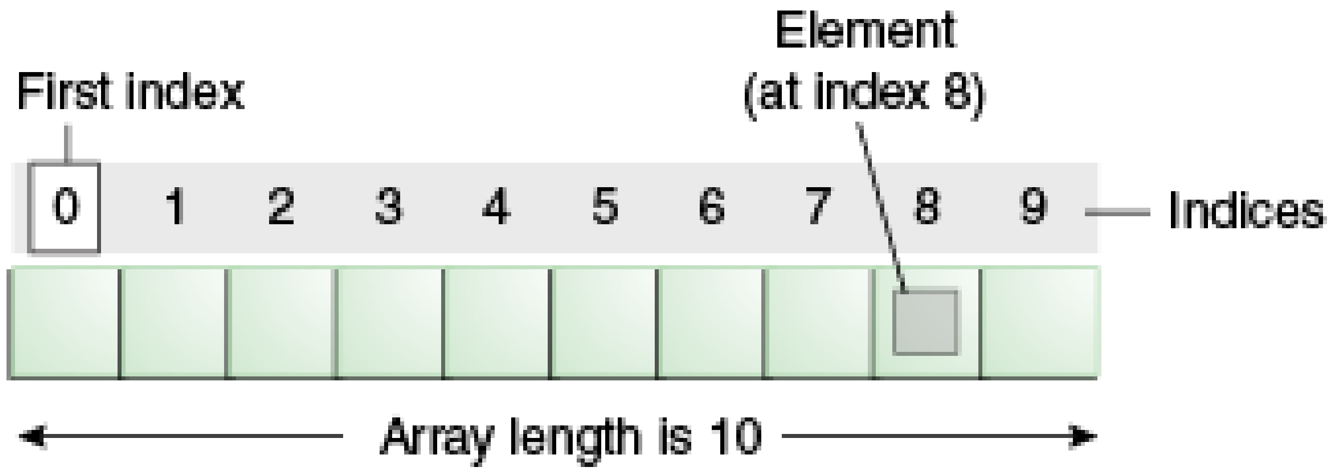
- the number of elements in the array is stored in the public field named length

```
double[] collection;
// collection is an array of double values


collection = new double[10];
// collection is an array of 10 double values

int n = collection.length;
// the public field length holds the number of elements
```

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Arrays

- the values in an array are called elements
- the elements can be accessed using a zero-based index (similar to lists and strings)



https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Arrays

- the elements can be accessed using a zero-based index (similar to lists and strings)

```
collection[0] = 100.0;
collection[1] = 100.0;
collection[2] = 100.0;
collection[3] = 100.0;
collection[4] = 100.0;
collection[5] = 100.0;
collection[6] = 100.0;
collection[7] = 100.0;
collection[8] = 100.0;
collection[9] = 100.0;  // set all elements to equal 100.0
collection[10] = 100.0; // ArrayIndexOutOfBoundsException
```

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Array vs ArrayList

- under most circumstances, you should use **`ArrayList`** instead of an array
  - however, arrays are a part of the Java language and it is important that you understand how to use them
- advantages of **`ArrayList`**
  - grows in size automatically when needed
  - provides many useful methods