EECS 2030 Advanced Object-Oriented Programming

S2023, Section A

Aggregation and Composition:
Collections as Fields (going deeper)

Aggregation and Composition

Composition implies ownership

- if the university disappears then all of its departments disappear
- a university is a composition of departments

Aggregation does *not* imply ownership

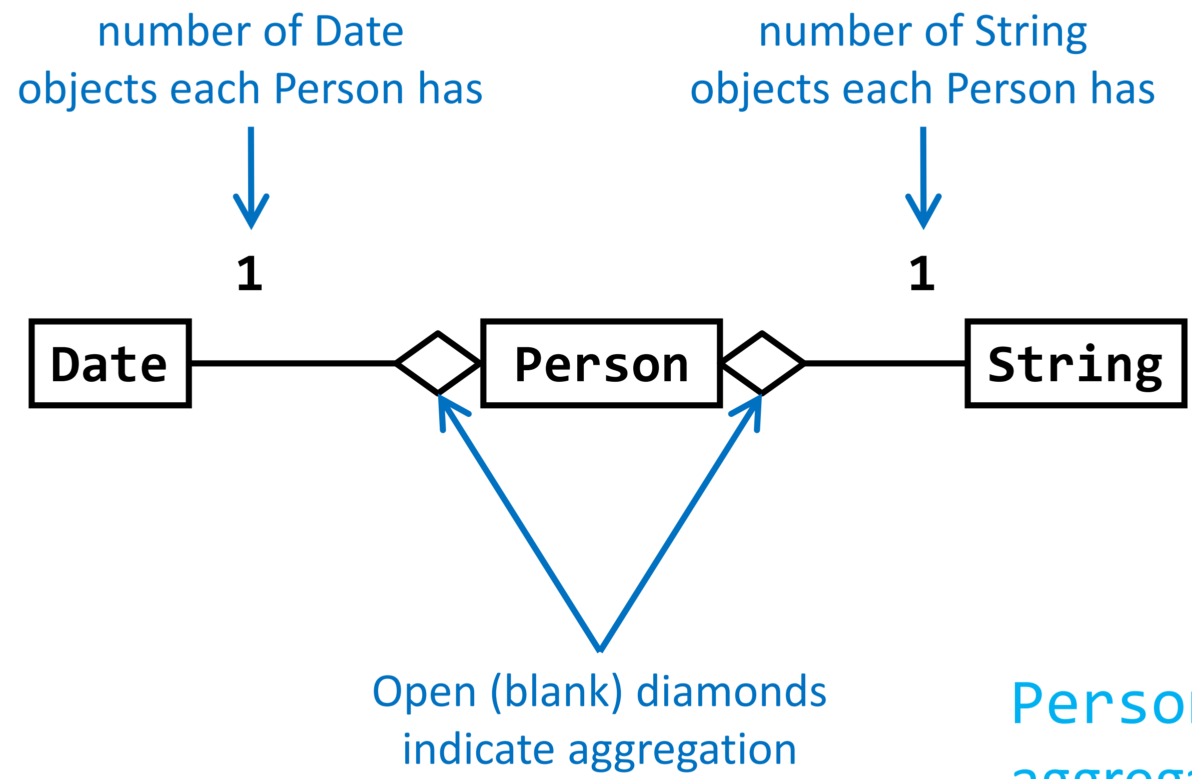
- if a department disappears then the professors do not disappear
- a department is an aggregation of professors

Aggregation

suppose a Person has a name and a date of birth

```
public class Person {
  private String name;
  private Date birthDate;
```

UML Class Diagram for Aggregation


number of Date objects each Person has
number of String objects each Person has
1
1
Date
Person
String
Open (blank) diamonds indicate aggregation

`Person` is an aggregation of `Date` and `String`

Composition

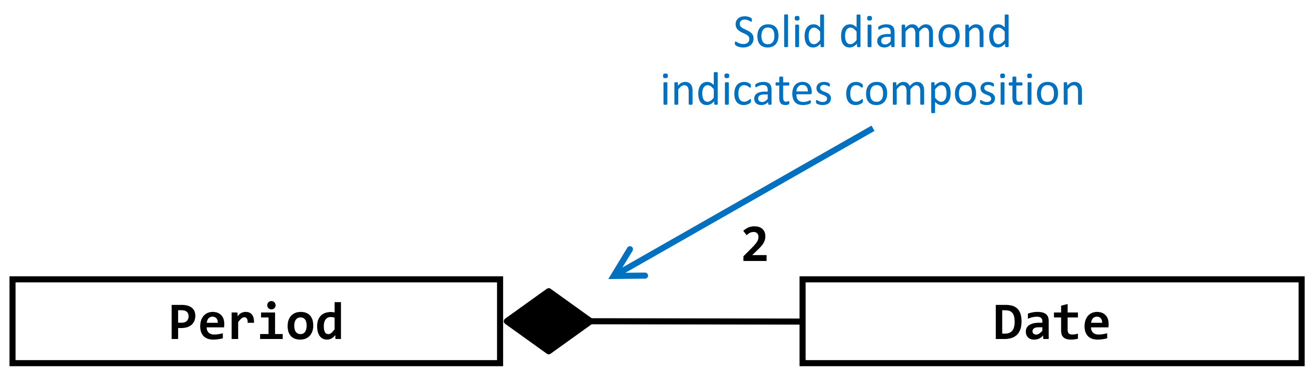
recall that an object of type X that is composed of an object of type Y means

- X has-a Y object and
- X owns the Y object

in other words

the **X** object has **exclusive** access to its **Y** object

Period Class



Period is a composition of two **Date** objects

Privacy Leaks

a privacy leak occurs when a class exposes a reference to a non-public field (that is not a primitive or immutable)

given a class **X** that is a composition of a **Y**

```
public class X {
  private Y y;
  // …
}
```

these are all examples of privacy leaks

```
public X(Y y) {
  this.y = y;
}
```

```
public X(X other) {
  this.y = other.y;
}
```

```
public Y getY() {
  return this.y;
}
```

```
public void setY(Y y) {
  this.y = y;
}
```

Collections as Fields: 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*

*triangles are easier to work with than more complex shapes

What Does a Collection Hold?

a collection holds references to instances

- it *does not* hold the instances

Normally, we should write *List* here

```
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);
```

100	`client` invocation
dates	200a
d1	500a
d2	600a
d3	700a
	...
200	`ArrayList` object
	500a
	600a
	700a

What does the following print?

```
ArrayList<Point2> pts = new ArrayList<Point2>();
Point2 p = new Point2(0., 0.);
pts.add(p);
p.x( 10.0 );
System.out.print(p);
System.out.println(", " + pts.get(0));
```

a. `(0.0, 0.0), (0.0, 0.0)`
b. `(0.0, 0.0), (10.0, 0.0)`
c. `(10.0, 0.0), (0.0, 0.0)`
d. `(10.0, 0.0), (10.0, 0.0)`

Question

Is an ArrayList<X> an aggregation of X or a composition of X?

Aggregation?

Composition?

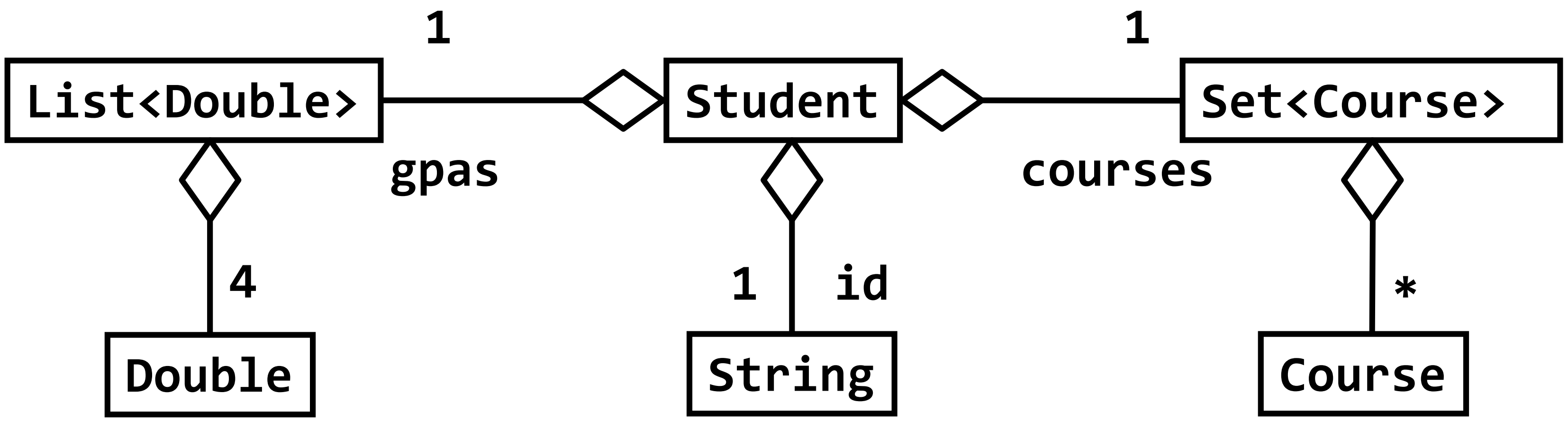
Neither?

Student Class (from notes)

Student has-a string id

Student has-a collection of yearly GPAs

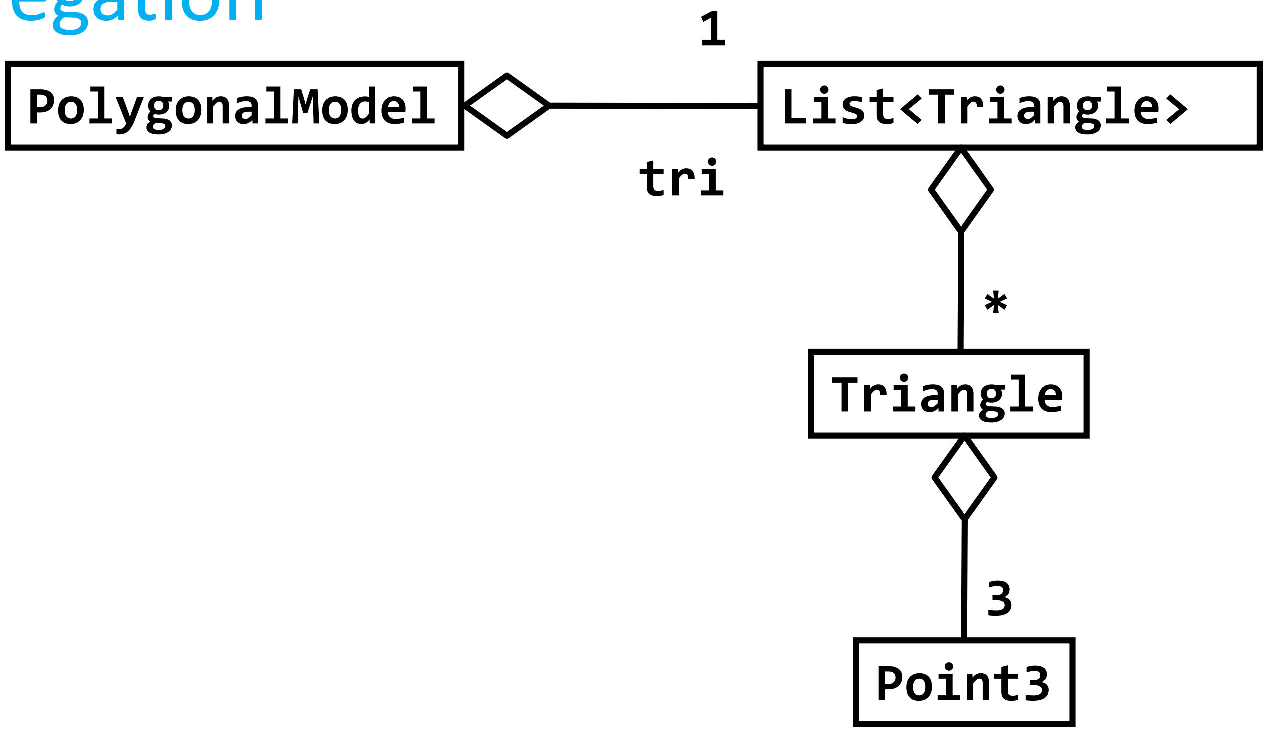
Student has-a collection of courses

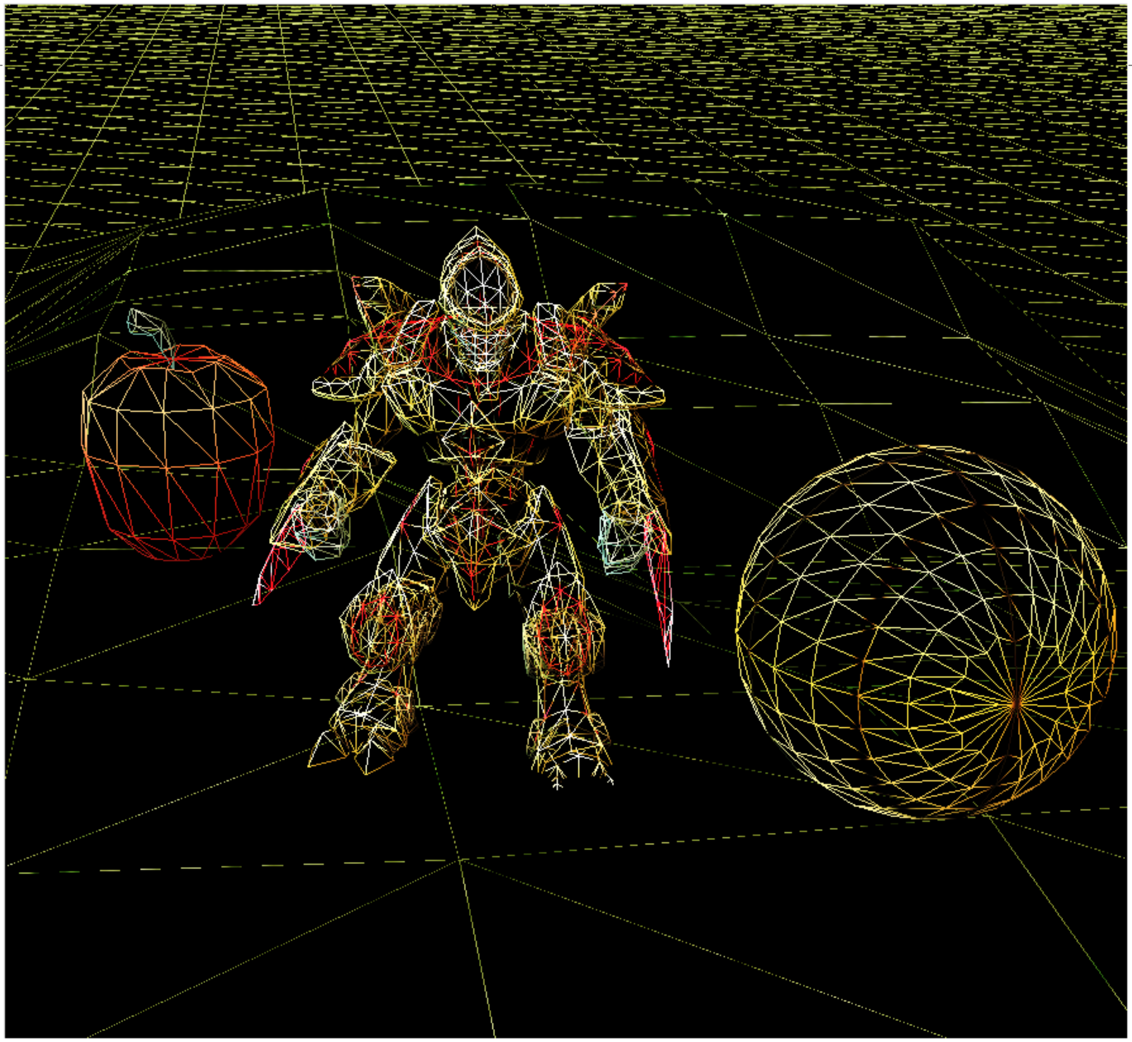


PolygonalModel Class

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 field 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
- **X**'s collection is an alias for another collection

PolygonalModel Copy Constructor 1

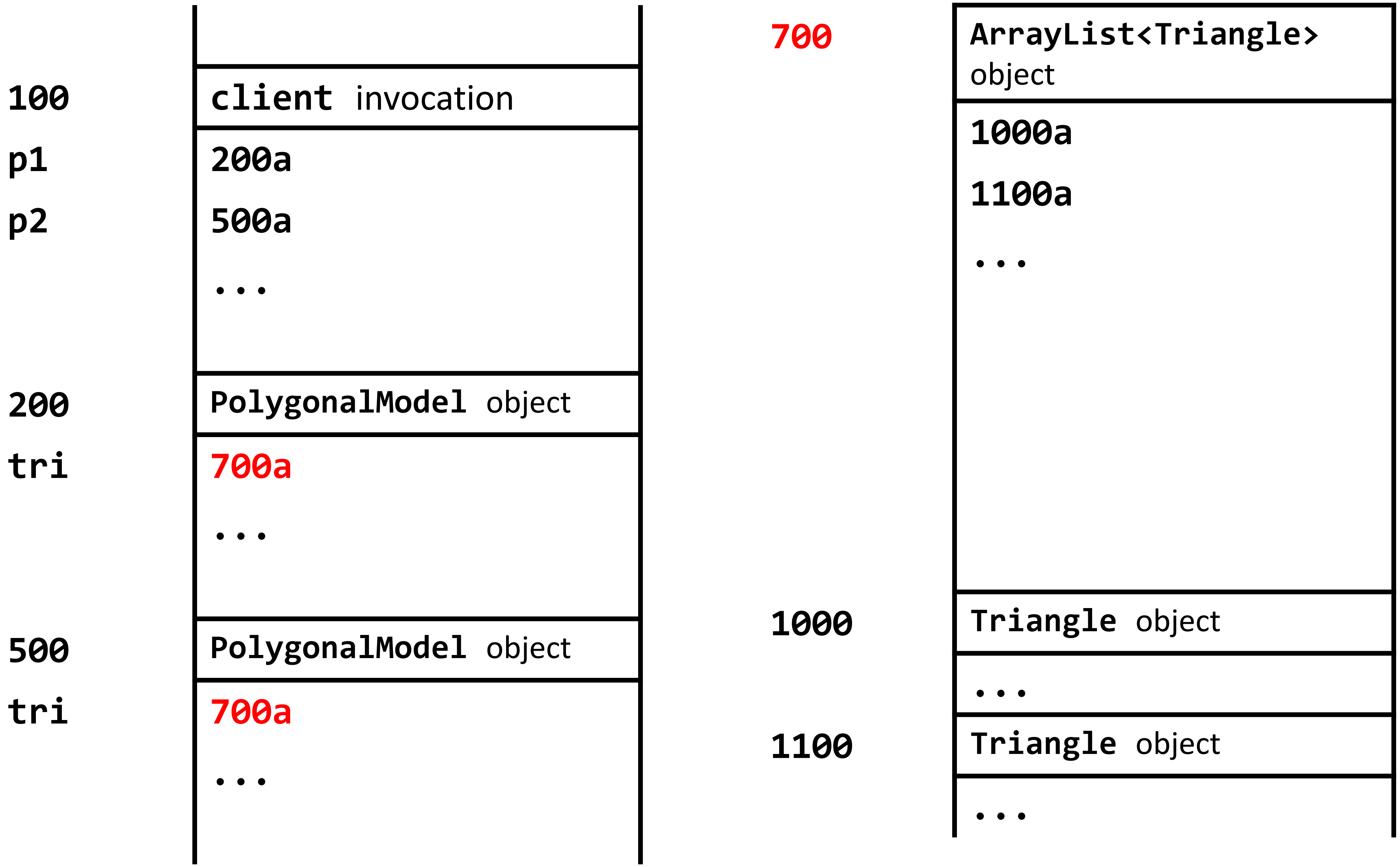
```
public PolygonalModel(PolygonalModel other) {
    // implements aliasing (sharing) with other
    //   PolygonalModel instances
    this.tri = other.tri;
}
```

alias: no new **List** created

alias: no new **List** created

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

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



Question

Suppose that the **`PolygonalModel`** copy constructor makes an alias of the list of triangles.
Suppose you have a **`PolygonalModel p1`** that has 100 **`Triangles`**. What does the following code print?

```
PolygonalModel p2 = new PolygonalModel(p1);
p2.clear();
System.out.print( p2.size() );
System.out.println( ", " + p1.size() );
```

a. `0, 0`
b. `0, 100`

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);
```

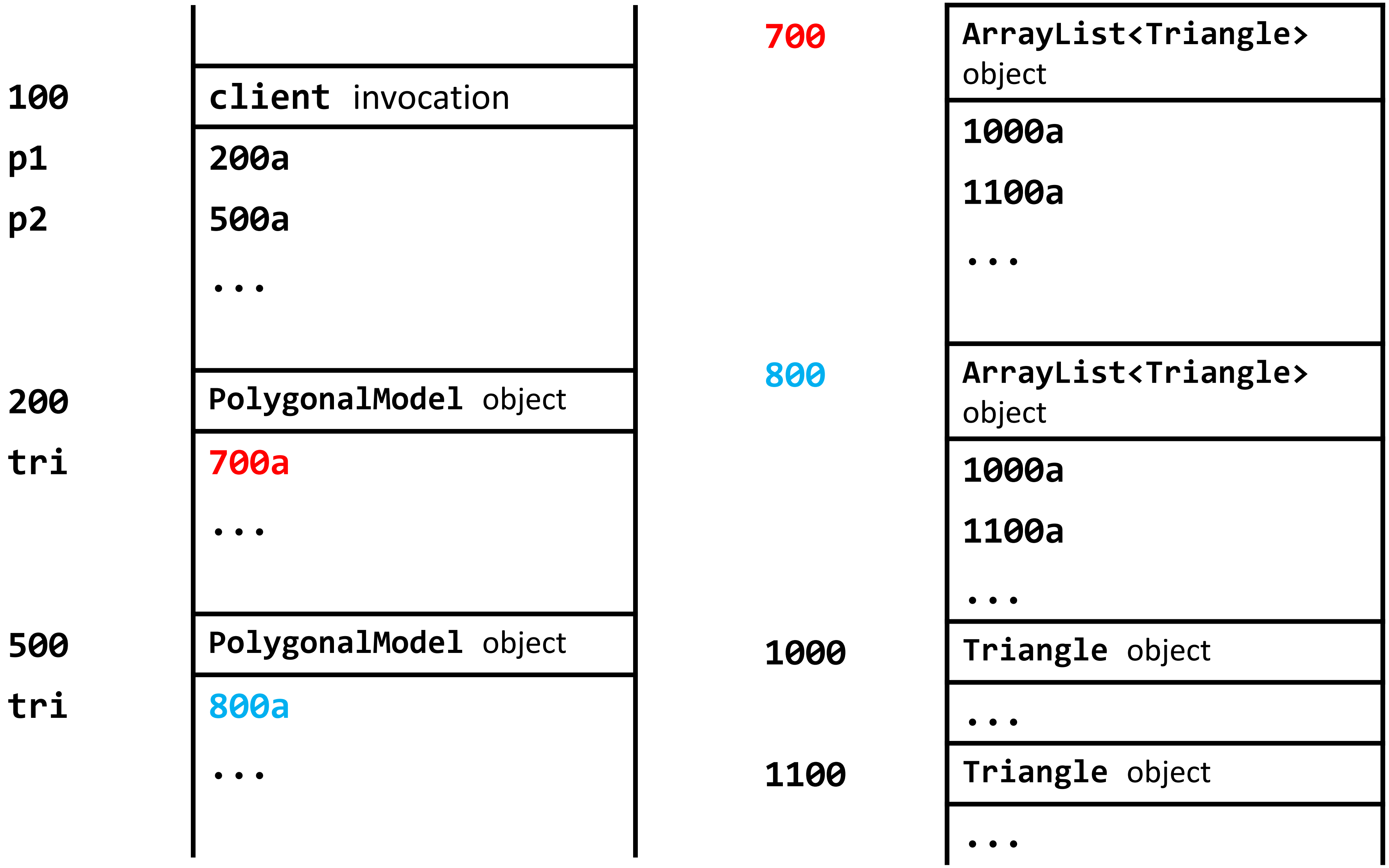
`List` and **`Set`** constructors that have a **`Collection`** as a parameter make a shallow copy of the **`Collection`**

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);
```



Question

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.print( p2.size() );
System.out.println( ", " + p1.size() );
```

a. `0, 0`
b. `0, 100`

Question

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);
```

a. false

b. true

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()));
}
```

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

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

PolygonalModel Copy Constructor 3

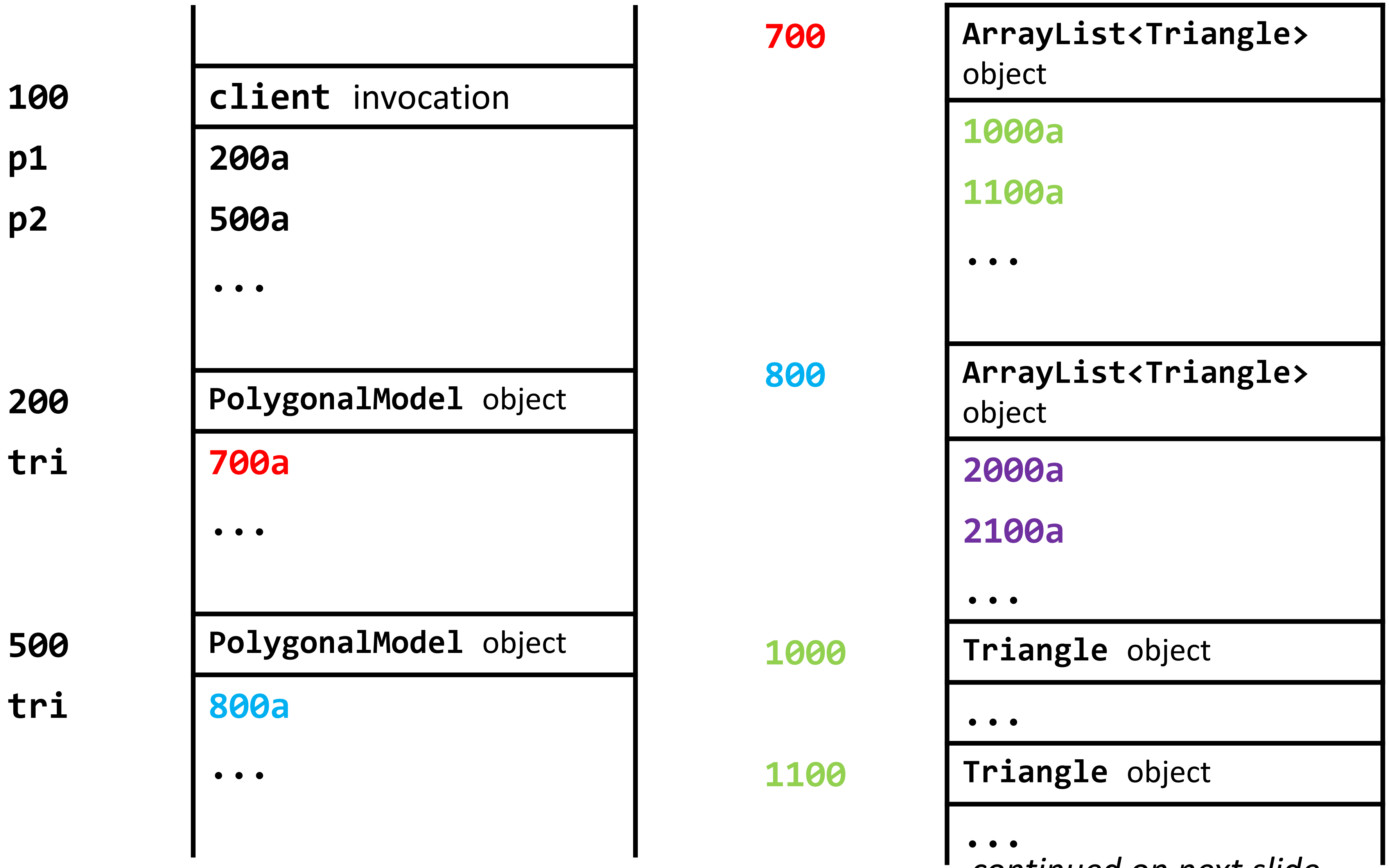
```
public PolygonalModel(PolygonalModel other) {


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

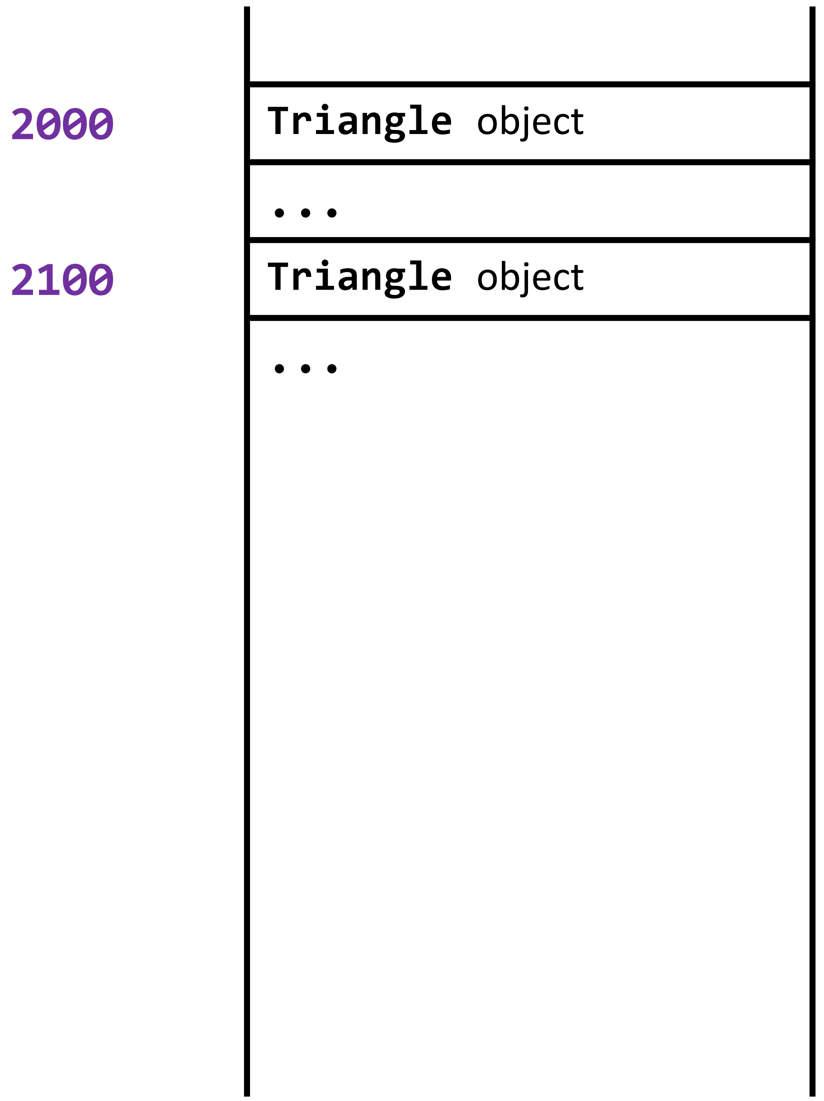
deep copy: new **`List`** created, and new **`Triangle`** objects created

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

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



continued on next slide


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

Question

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() );
```

A. **`0, 0`**

B. **`0, 100`**

Question

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.print( t1 == t2 );
System.out.println( ", " + t1.equals(t2) );
```

A. `false, true`

B. `true, true`