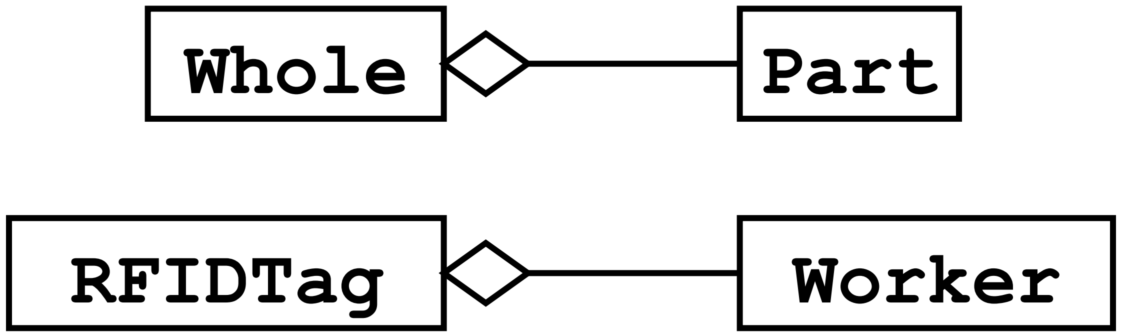
Aggregation

Definition

- Represents a "has-a" relationship between two classes
- Class *W* is an "aggregate" if it has an attribute of type *P* and *P* is <u>**not**</u> a primitive type
- Attribute *P* is called the "aggregated part", "part", "aggregated component", or just "component"

UML Diagram



- The **`Whole`** class has an attribute of type **`Part`**
- The class **`Whole`** aggregates **`Part`**

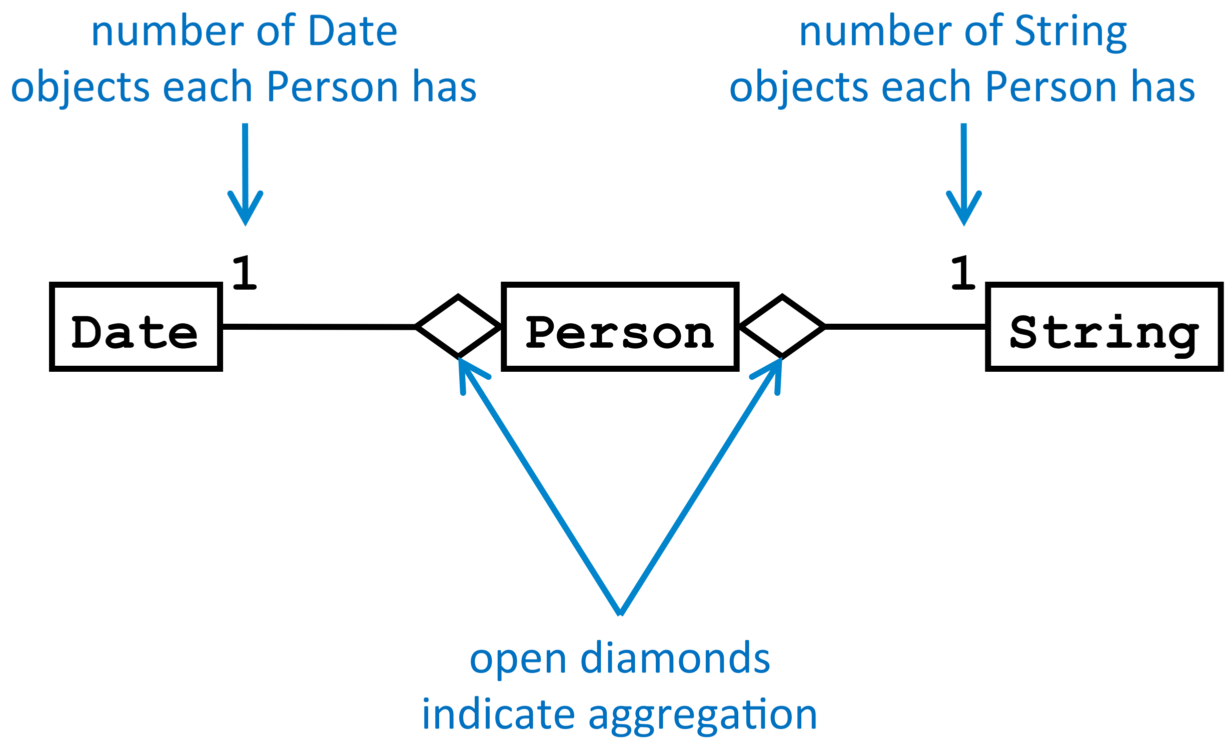
- The **`RFIDTag`** class has a **`Worker`** attribute
- The class **`RFIDTag`** aggregates **`Worker`**

Aggregation Example

- Suppose a **`Person`** has a name and a date of birth

```
public class Person
{
  private String name;
  private Date birthDate;
  public Person(String name, Date birthDate) {
    this.name = name;
    this.birthDate = birthDate;
  }
  public Date getBirthDate()
  {
    return birthDate;
  }
}
```

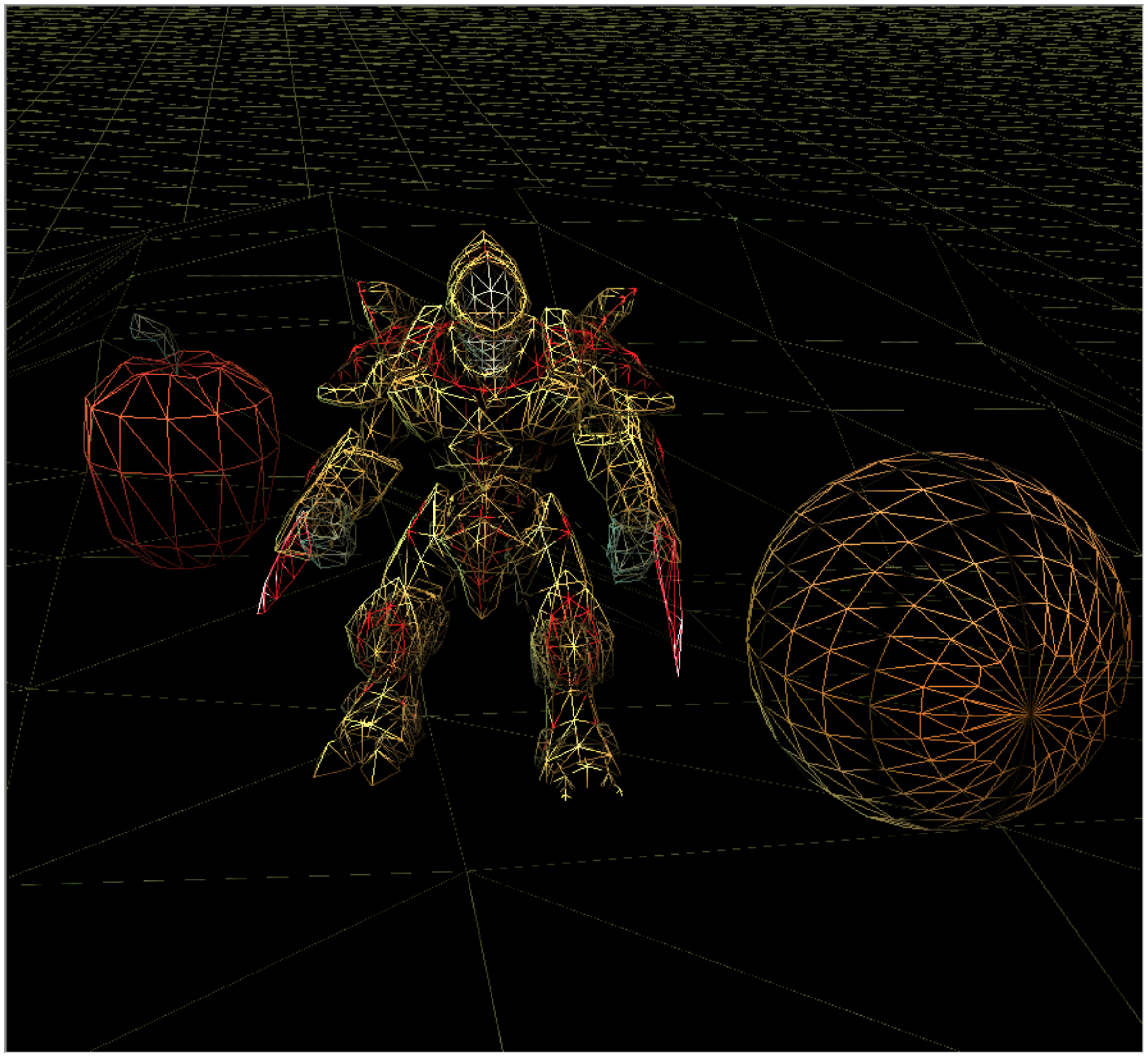
UML Class Diagram for Aggregation


number of Date objects each Person has
number of String objects each Person has
1
1
Date
Person
String
open diamonds indicate aggregation

Another Aggregation Example

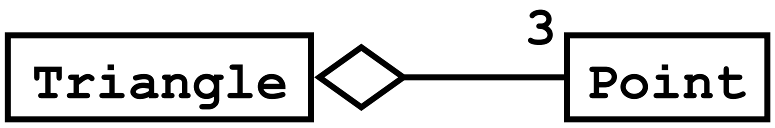
- 3D videogames use models that are a three dimensional representations of geometric data
  - Models may be represented by:
    - Three-dimensional points (particle systems)
    - Simple polygons (triangles, quadrilaterals)
    - Smooth, continuous surfaces (splines, parametric surfaces)
    - An algorithm (procedural models)

- Rendering the objects to the screen usually results in drawing triangles
  - Graphics cards have specialized hardware that does this very fast



Aggregation Example

- A **`Triangle`** has 3 three-dimensional **`Point`**s



Triangle
`+ Triangle(Point, Point, Point)`

`+ getA() : Point`
`+ getB() : Point`
`+ getC() : Point`
`+ setA(Point) : void`
`+ setB(Point) : void`
`+ setC(Point) : void`

Point
`+ Point(double, double, double)`

`+ getX() : double`
`+ getY() : double`
`+ getZ() : double`
`+ setX(double) : void`
`+ setY(double) : void`
`+ setZ(double) : void`

Triangle (attributes and constructor)

```
public class Triangle {
  private Point pA;
  private Point pB;
  private Point pC;

  public Triangle(Point a, Point b, Point c) {
    this.pA = a;
    this.pB = b;
    this.pC = c;
  }
}
```

Triangle (accessors)

```
 public Point getA() {
   return this.pA;
}

public Point getB() {
   return this.pB;
}

public Point getC() {
   return this.pC;
}
```

Triangle (mutators)

```
public void setA(Point p) {
    this.pA = p;
  }

  public void setB(Point p) {
    this.pB = p;
  }

  public void setC(Point p) {
    this.pC = p;
  }
```

Triangle Aggregation

- Implementing **`Triangle`** is very easy
- Attributes (3 **`Point`** references)
  - References to existing objects provided by the user of the class
- Accessors
  - Give users of a class access to an attribute
- Mutators
  - Allows users of a class to change an attribute
- We say that the **`Triangle`** attributes are *aliases*
  (i.e., references to the Point objects)

Client code – the user of a class

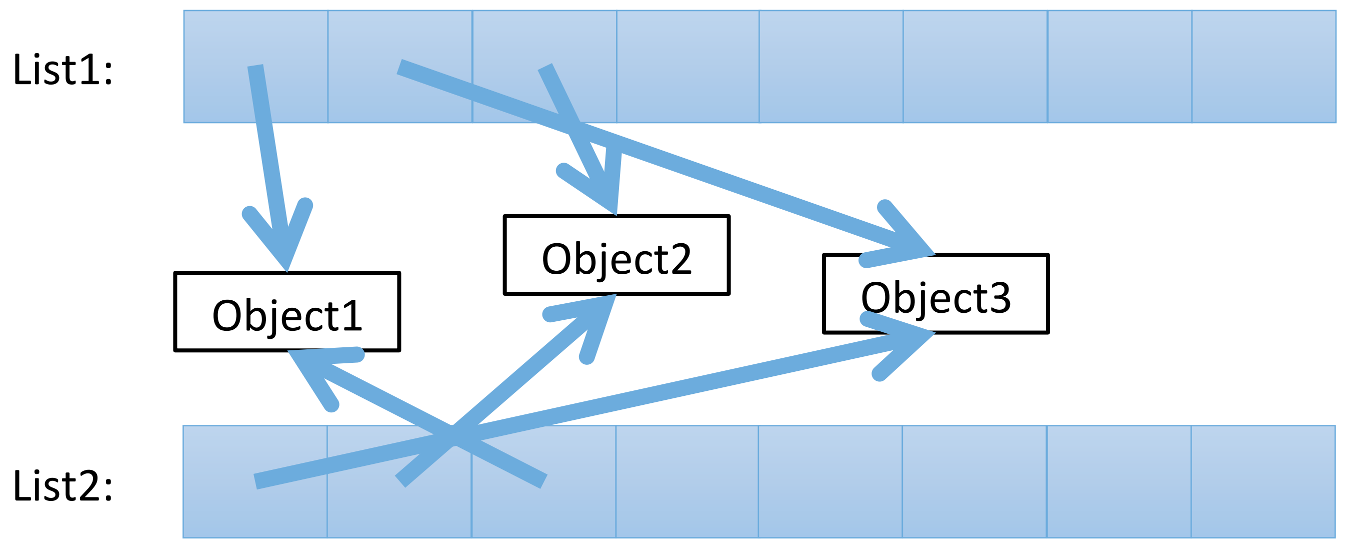
```
Point a = new Point(-1.0, -1.0, -3.0);
Point b = new Point(0.0, 1.0, -3.0);
Point c = new Point(2.0, 0.0, -3.0);
Triangle tri = new Triangle(a, b, c);
```

Collections as Attributes

- Often you will want to implement a class that has-a collection as an attribute
  - A university has-a collection of faculties and each faculty has-a collection of schools and departments
  - A molecule has-a collection of atoms
  - A person has-a collection of acquaintances
  - A student has-a collection of GPAs and has-a collection of courses
  - A polygonal model has-a collection of triangles

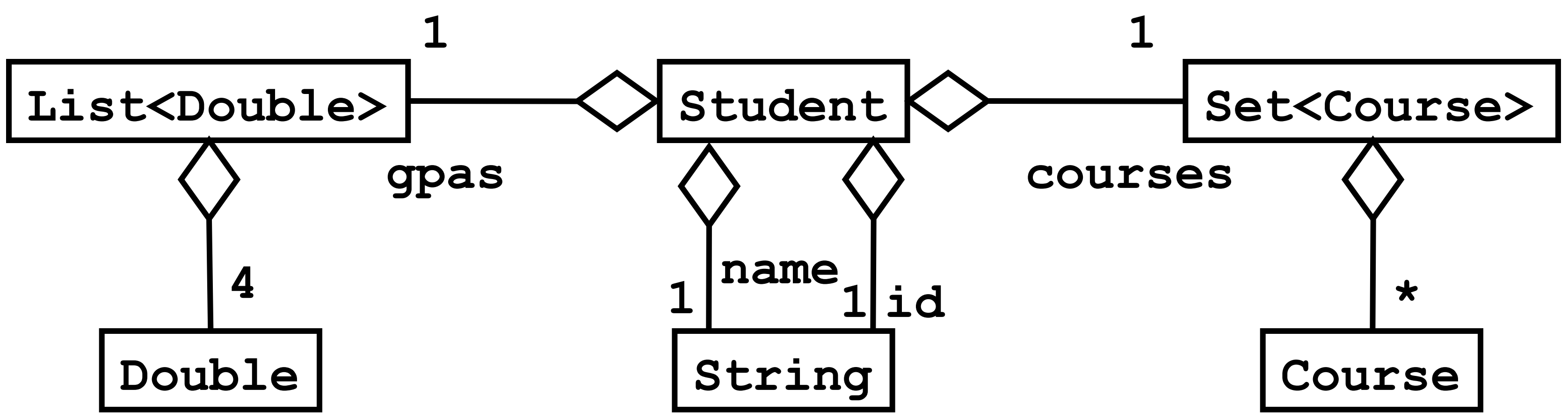
What Does a Collection Hold?

- A collection holds references (i.e., memory addresses) to its elements (i.e., objects)
  - It does not hold the elements themselves
  - Allows an object to be held by multiple collections



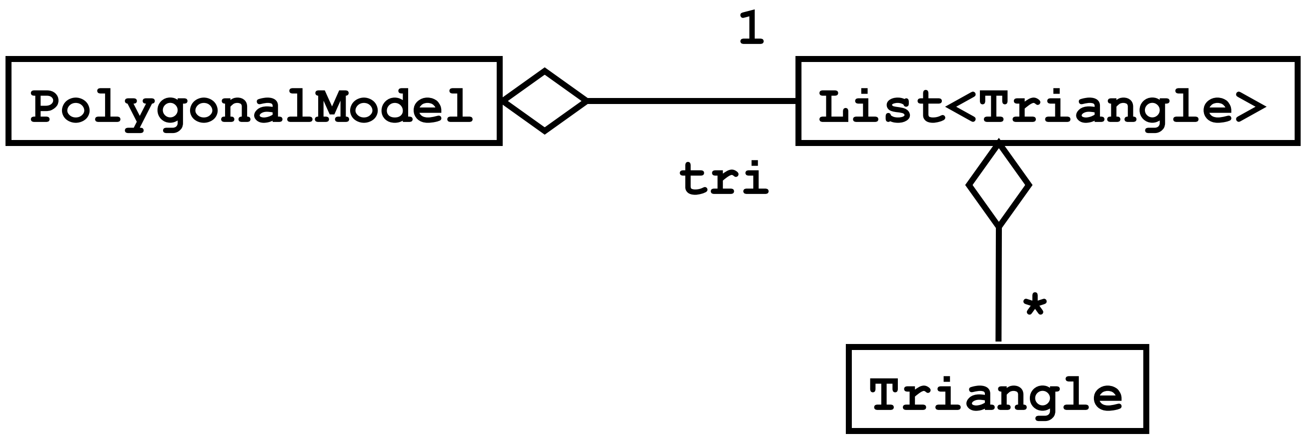
Student Class

- A Student has-a:
  - String name and id
  - Collection of yearly GPAs
  - Collection of courses



PolygonalModel Class

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



PolygonalModel

```
public class PolygonalModel
{
  private List<Triangle> tri;

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


  …
```