Monday Nov. 19

Lecture 20

Abstract vs. Concrete Implementations

```
Polygon p = new Polygon();
p.getArea();
```

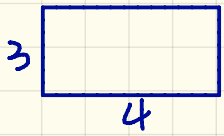
abstract double getArea()

Polygon

double[] sides
void grow()
double getPerimeter()

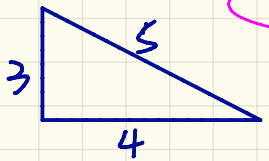
Rectangle

double getArea()
w * l



Triangle

double getArea()
$\sqrt{s(s-a)(s-b)(s-c)}$



$\sqrt{6 \cdot 1 \cdot 3 \cdot 2} = 6$

Abstract Class and descendants

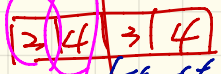
Polygon p;

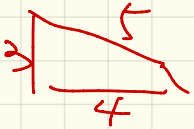
p = ~~new~~ Polygon(..);
p = new Triangle(..);
p = new Rectangle(..);

```
public abstract class Polygon {
  double[] sides;
  Polygon(double[] sides) { this.sides = sides; }
  void grow() {
    for(int i = 0; i < sides.length; i ++) { sides[i] ++; }
  }
  double getPerimeter() {
    double perimeter = 0;
    for(int i = 0; i < sides.length; i ++) {
      perimeter += sides[i];
    }
    return perimeter;
  }
  abstract double getArea();
}
```

super

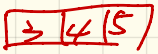
anonymous object





extends

extends




```
public class Rectangle extends Polygon {
  Rectangle(double length, double width) {
    super(new double[4]);
    sides[0] = length; sides[1] = width;
    sides[2] = length; sides[3] = width;
  }
  double getArea() { return sides[0] * sides[1]; }
}
```

```
public class Triangle extends Polygon {
  Triangle(double side1, double side2, double side3) {
    super(new double[3]);
    sides[0] = side1; sides[1] = side2; sides[2] = side3;
  }
  double getArea() {
    /* Heron's formula */
    double s = getPerimeter() * 0.5;
    double area = Math.sqrt(
      s * (s - sides[0]) * (s - sides[1]) * (s - sides[2]));
    return area;
  }
}
```

Rectangle (3, 4)

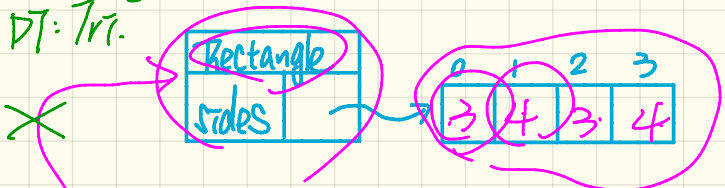
Polymorphic Collection of Polygons

```java
public abstract class Polygon {
  double[] sides;
  Polygon(double[] sides) { this.sides = sides; }
  void grow() {
    for(int i = 0; i < sides.length; i ++) { sides[i] ++; }
  }
  double getPerimeter() {
    double perimeter = 0;
    for(int i = 0; i < sides.length; i ++) {
      perimeter += sides[i];
    }
    return perimeter;
  }
  abstract double getArea();
}
```

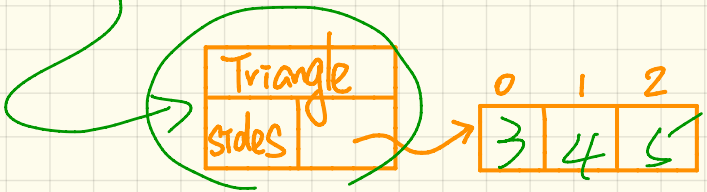
```java
Polygon p;
p = new Rectangle(3, 4); /* polymorphism */
System.out.println(p.getPerimeter()); /* 14.0 */
System.out.println(p.getArea()); /* 12.0 */
p = new Triangle(3, 4, 5); /* polymorphism */
System.out.println(p.getPerimeter()); /* 12.0 */
System.out.println(p.getArea()); /* 6.0 */
```

DT: Rec.

DT: Tri.



Polygon p

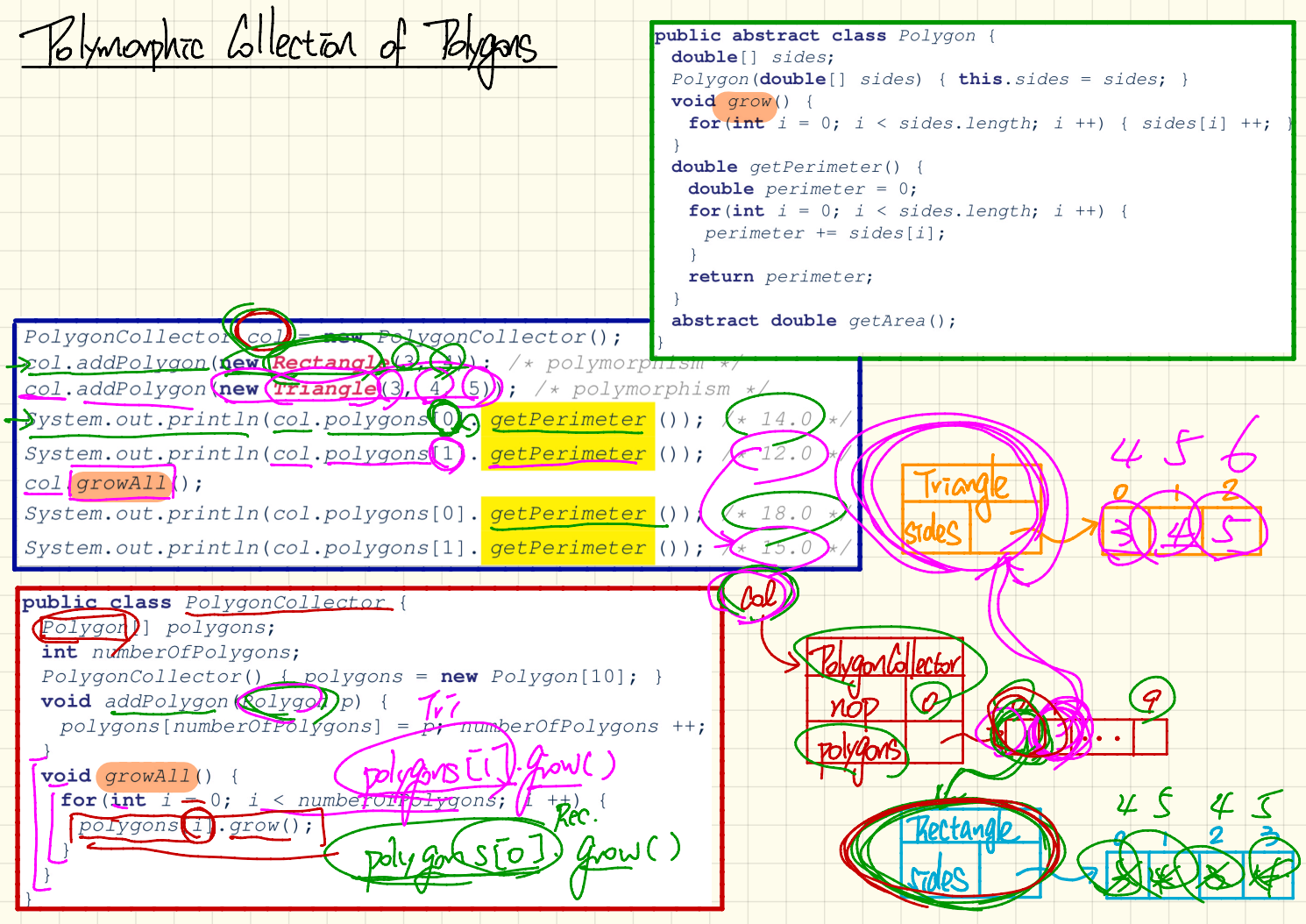


Polymorphic Collection of Polygons

```
public abstract class Polygon {
  double[] sides;
  Polygon(double[] sides) { this.sides = sides; }
  void grow() {
    for(int i = 0; i < sides.length; i ++) { sides[i] ++; }
  }
  double getPerimeter() {
    double perimeter = 0;
    for(int i = 0; i < sides.length; i ++) {
      perimeter += sides[i];
    }
    return perimeter;
  }
  abstract double getArea();
}
```

```
PolygonCollector col = new PolygonCollector();
col.addPolygon(new Rectangle(3, 4)); /* polymorphism */
col.addPolygon(new Triangle(3, 4, 5)); /* polymorphism */
System.out.println(col.polygons[0].getPerimeter()); /* 14.0 */
System.out.println(col.polygons[1].getPerimeter()); /* 12.0 */
col.growAll();
System.out.println(col.polygons[0].getPerimeter()); /* 18.0 */
System.out.println(col.polygons[1].getPerimeter()); /* 15.0 */
```

```
public class PolygonCollector {
  Polygon[] polygons;
  int numberOfPolygons;
  PolygonCollector() { polygons = new Polygon[10]; }
  void addPolygon(Polygon p) {
    polygons[numberOfPolygons] = p; numberOfPolygons ++;
  }
  void growAll() {
    for(int i = 0; i < numberOfPolygons; i ++) {
      polygons[i].grow();
    }
  }
}
```

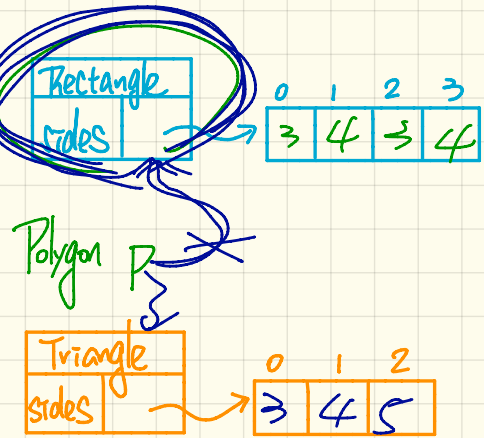


Polymorphic Return Values of Polygons

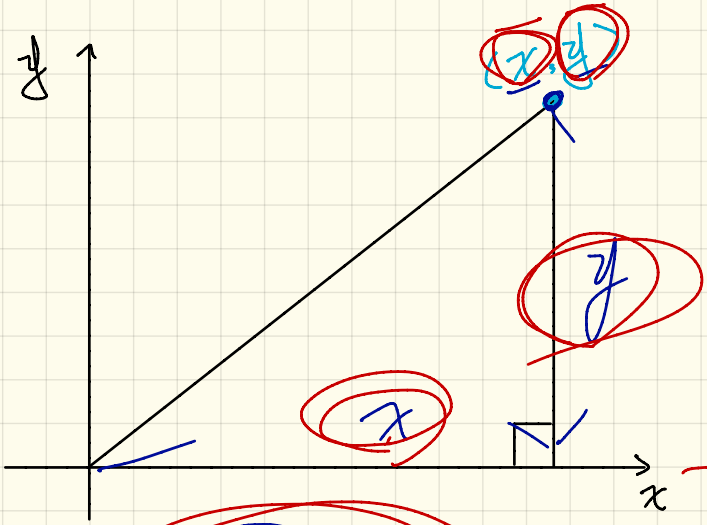
```
PolygonConstructor con = new PolygonConstructor();
double[] recSides = {3, 4, 3, 4}; Polygon p = con.getPolygon(recSides);
System.out.println(p instanceof Polygon);
System.out.println(p instanceof Rectangle);
System.out.println(p instanceof Triangle);
System.out.println(p.getPerimeter()); /* 14.0 */
System.out.println(p.getArea()); /* 12.0 */
con.grow(p);
System.out.println(p.getPerimeter()); /* 18.0 */
System.out.println(p.getArea()); /* 20.0 */
double[] triSides = {3, 4, 5}; p = con.getPolygon(triSides);
System.out.println(p instanceof Polygon);
System.out.println(p instanceof Rectangle);
System.out.println(p instanceof Triangle);
System.out.println(p.getPerimeter()); /* 12.0 */
System.out.println(p.getArea()); /* 6.0 */
con.grow(p);
System.out.println(p.getPerimeter()); /* 15.0 */
System.out.println(p.getArea()); /* 9.921 */
```

```
public abstract class Polygon {
  double[] sides;
  Polygon(double[] sides) { this.sides = sides; }
  void grow() {
    for(int i = 0; i < sides.length; i ++) { sides[i] ++; }
  }
  double getPerimeter() {
    double perimeter = 0;
    for(int i = 0; i < sides.length; i ++) {
      perimeter += sides[i];
    }
    return perimeter;
  }
  abstract double getArea();
}
```

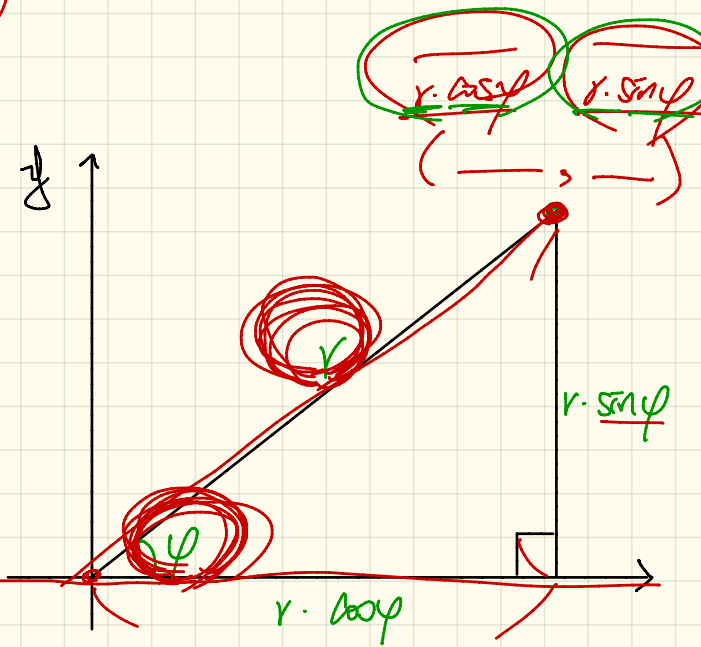
```
public class PolygonConstructor {
  Polygon getPolygon(double[] sides) {
    Polygon p = null;
    if(sides.length == 3) {
      p = new Triangle(sides[0], sides[1], sides[2]);
    }
    else if(sides.length == 4) {
      p = new Rectangle(sides[0], sides[1]);
    }
    return p;
  }
  void grow(Polygon p) { p.grow(); }
}
```



Two Representations of a 2D Point



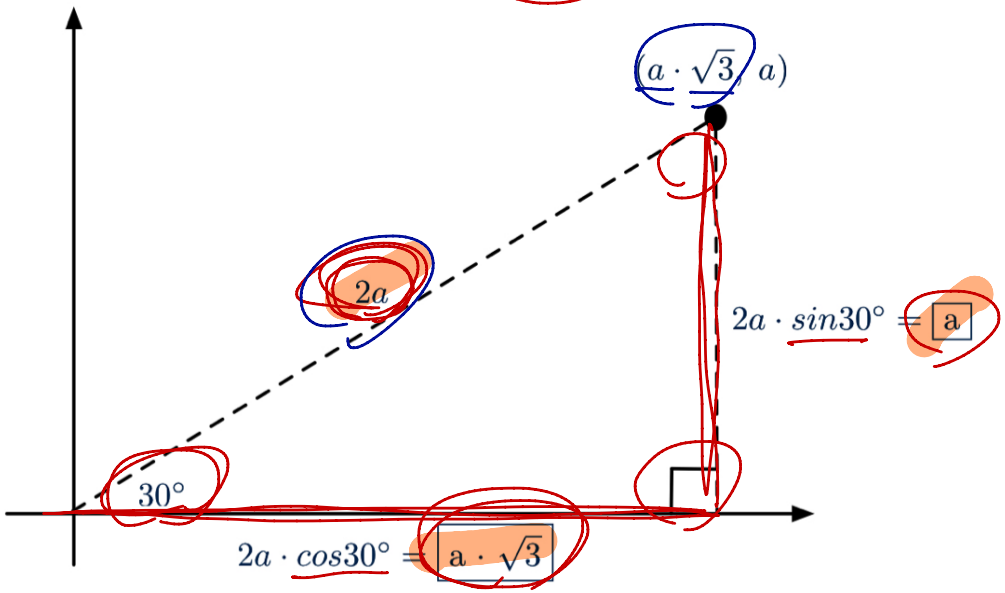
Cartesian



Polar

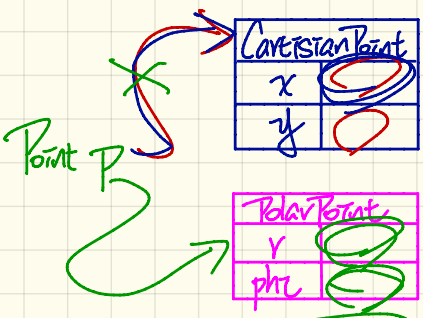
Cartesian vs. Polar: Example

Recall: $sin30° = \frac{1}{2}$ and $cos30° = \frac{1}{2} \cdot \sqrt{3}$



We consider the same point represented differently as:

- $r = 2a$, $\psi = 30°$ [polar system]
- $x = 2a \cdot cos30° = a \cdot \sqrt{3}$, $y = 2a \cdot sin30° = a$ [cartesian system]




```java
interface Point {
    double getX();
    double getY();
}
```

implements

implements

```java
public class CartesianPoint implements Point {
  double x;
  double y;
  CartesianPoint(double x, double y) {
    this.x = x;
    this.y = y;
  }
  public double getX() { return x; }
  public double getY() { return y; }
}
```

```java
public class PolarPoint implements Point {
  double phi;
  double r;
  public PolarPoint(double r, double phi) {
    this.r = r;
    this.phi = phi;
  }
  public double getX() { return Math.cos(phi) * r; }
  public double getY() { return Math.sin(phi) * r; }
}
```

```java
double A = 5;
double X = A * Math.sqrt(3);
double Y = A;
Point p;
p = new CartesianPoint(X, Y); /* polymorphism */
print("(" + p.getX() + ", " + p.getY() + ")");
p = new PolarPoint(2 * A, Math.toRadians(30));
print("(" + p.getX() + ", " + p.getY() + ")");
```

DT: CP

DT: PP