Lecture 6

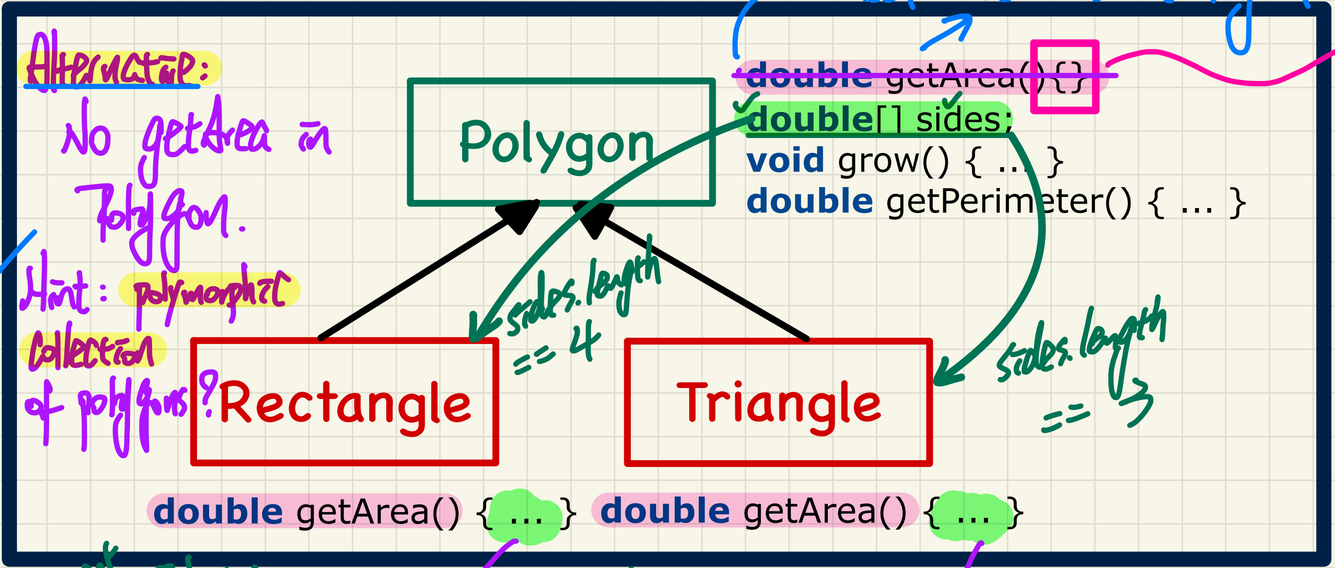
Part A

Abstract Classes

Abstract Implementation vs. Concrete Implementation

delayed/deferred to subclasses

Ideal: use abstract keyword



```
~~double getArea(){}~~
double[] sides;
void grow() { ... }
double getPerimeter() { ... }
```

Rectangle:

```
double getArea() { ... }
```

Triangle:

```
double getArea() { ... }
```

→ Empty Implementation

∵ at this level, we don't know how to calculate the area

Alternative: No getArea in Polygon.

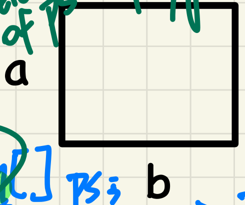
Hint: polymorphic collection of polygons?

not appropriate

sides.length == 4

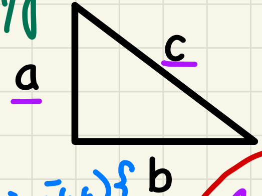
sides.length == 3

ST of each element of ps: Polygon

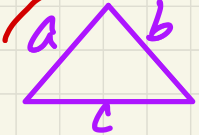


$a * b$

ST: Polygon



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



```
Polygon[] ps;
double total = 0;
for(int i=0; i<nop; i++){
  total += ps[i].getArea(); ✗
}
```

not compiling ∵ getArea not expected on Polygon.

Abstract Class vs. Concrete Descendants

```
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();
}
```

At least one method is abstract

implementation delayed to the subclasses

rather than {}

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]; }
}
```

0 1 2 3

3 4 3 4

*

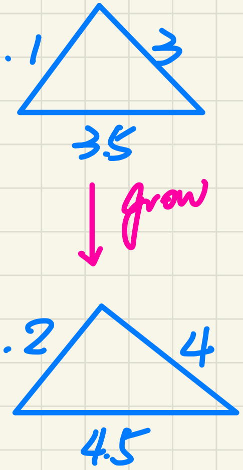
no longer abstract

```
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;
  }
}
```

0 1 2

3 4 5

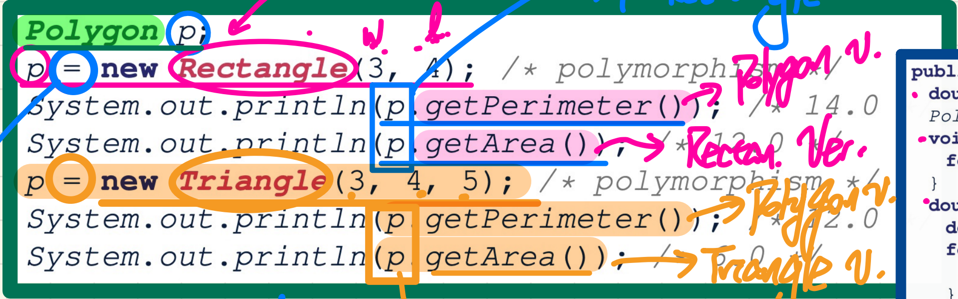
static method



Polymorphic Assignments of Polygons

DT: Rectangle.

p instanceof Rectangle ✓ ✗



```
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 */
```

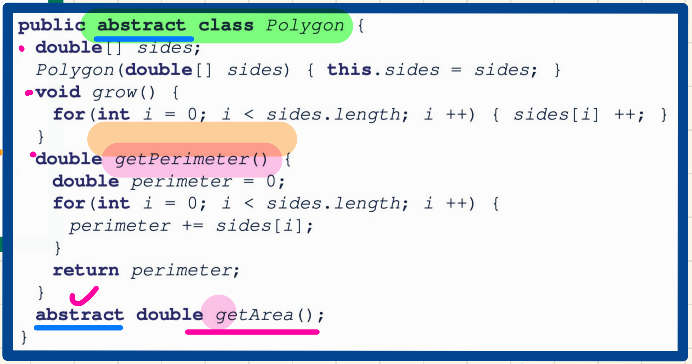
Polygon V.

Rectan. Ver.

Polygon V.

Triangle V.

DT: Triangle

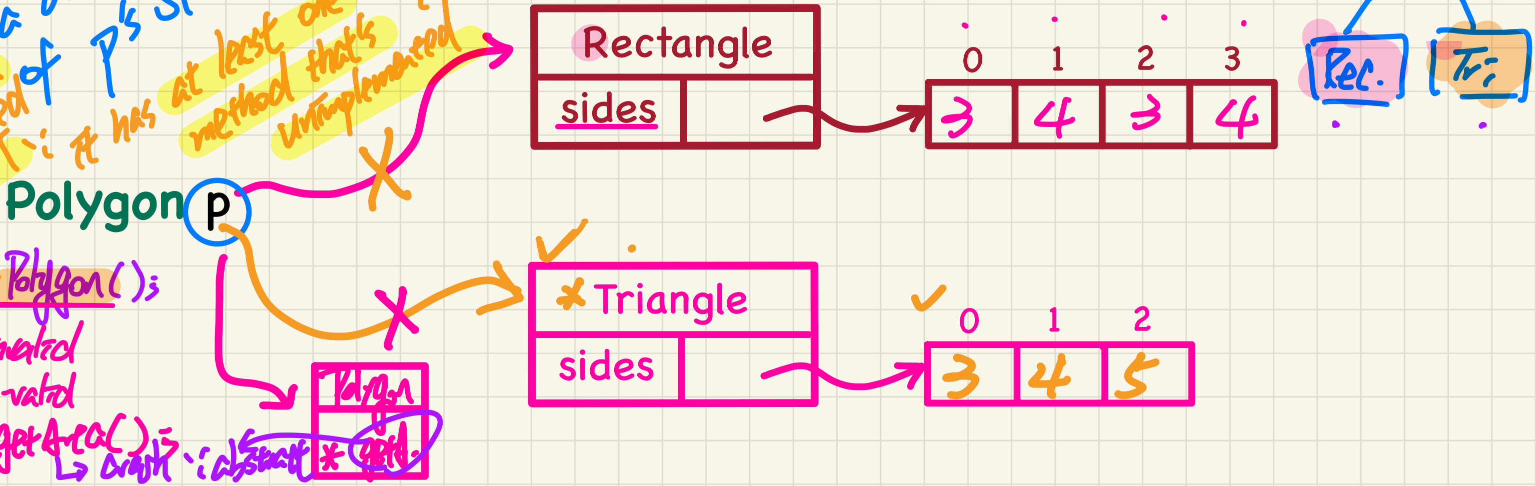


```
public abstract class Polygon {
  double[] sides;
  Polygon(double[] sides) { this.sides = sides; }
  void grow() {
    for(int i = 0; i < sides.length; i ++) { sides[i] ++; }
  }
  double getPerimeter() {
    double perimeter = 0;
    for(int i = 0; i < sides.length; i ++) {
      perimeter += sides[i];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

valid?

Yes! ∵ DT Rec. is a descendant class of p's ST (Polygon).

abstract class cannot be used as a DT ∵ it has at least one method that's unimplemented (abstract)



P = new Polygon();

X invalid

Assume valid

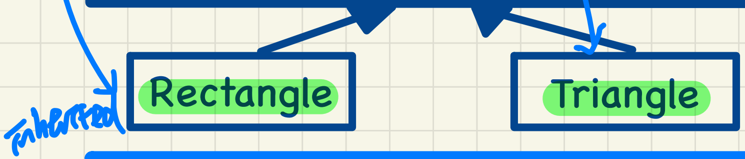
↳ p.getArea();

↳ crash ∵ abstract

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();
}
```

Inherited



DT: Rec.

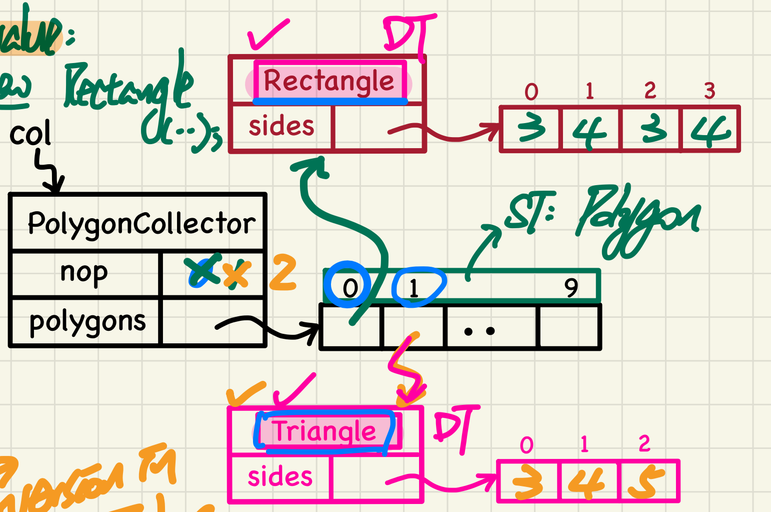
```
PolygonCollector col = new PolygonCollector();
col.addPolygon(new Rectangle(3, 4)); /* polymorphism
col.addPolygon(new Triangle(3, 4, 5)); /* polymorphis
System.out.println(col.polygons[0]. getPerimeter ());
System.out.println(col.polygons[1]. getPerimeter ());
col.growAll();
System.out.println(col.polygons[0].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

DT: Tri.

version of Polygon

Call by value:

P = new Rectangle(..);



```
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();
    }
  }
}
```

i

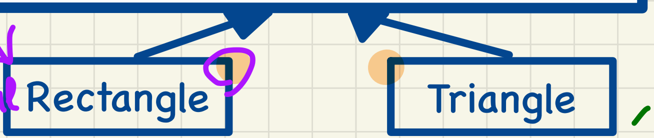
0 DT: Rec. polygons[0].grow()

1 DT: Tri. polygons[1].grow()

version of Polygon

Polymorphic Return Type 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();
}
```



ST of return value

```java
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(); }
}
```

valid ∵ expression to return (p) has ST Polygon, which is a descendant of RT Polygon.

call by value

```java
PolygonConstructor con = new PolygonConstructor();
double[] recSides = {3, 4, 3, 4}; 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 */
```

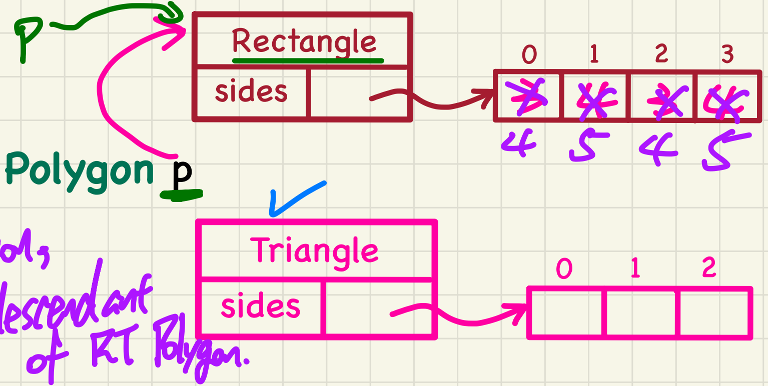
valid ∵ RHS's ST is a descendant of p's ST.

ST: Polygon

Polygon v.

Rec. ver.

DT: Rec.



Lecture 6

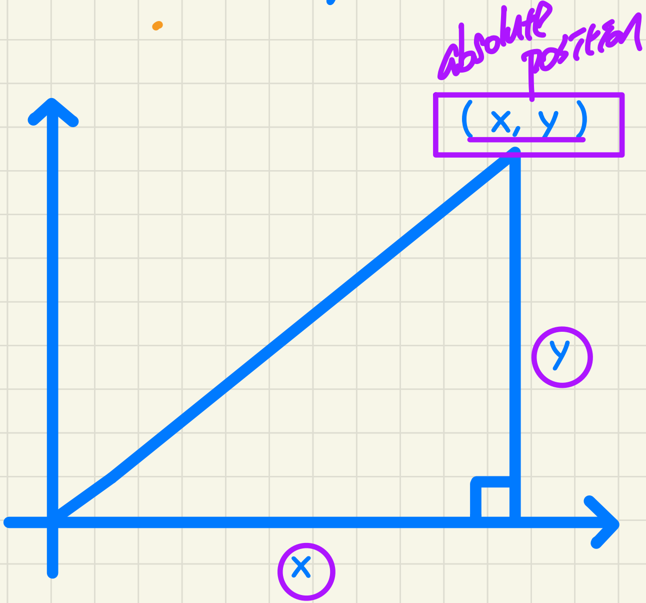
Part B

Interfaces

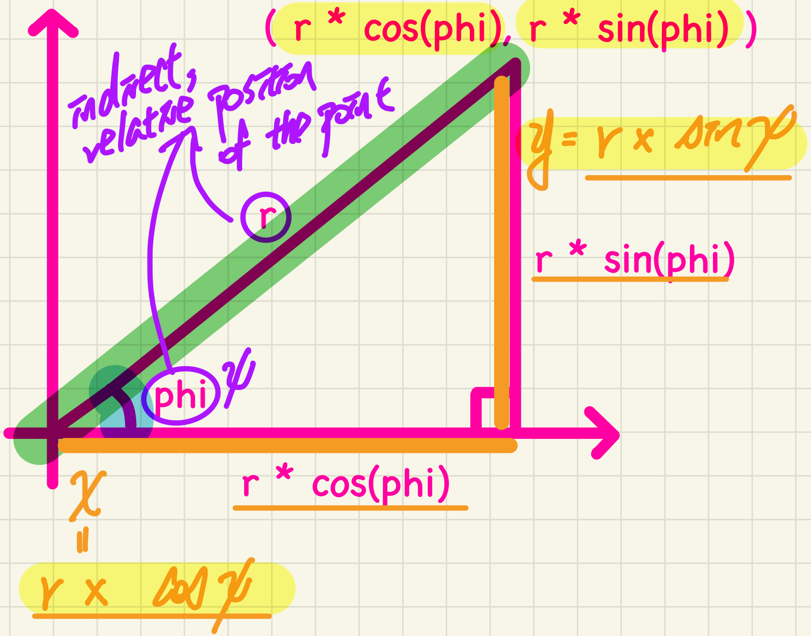
Representations of 2-D Points: Cartesian vs. Polar

Goal: Dynamically, switch between two systems seamlessly.

Cartesian System



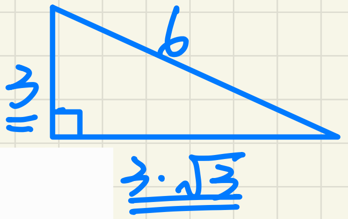
Polar System


(r * cos(phi), r * sin(phi))
indirect, relative position of the point
r
phi
φ
y = r x sin φ
r * sin(phi)
r * cos(phi)
x
=
r x cos φ

Example: Cartesian vs. Polar

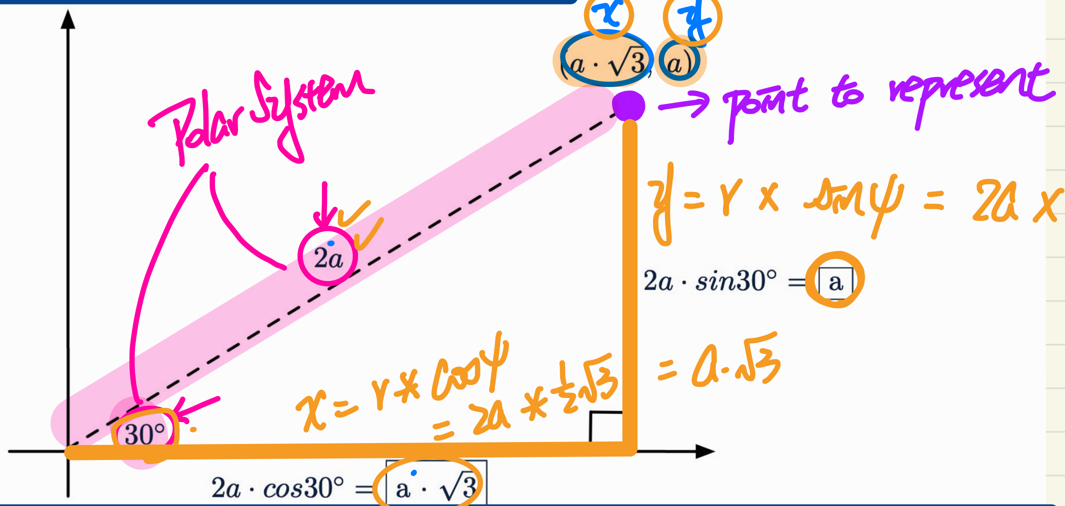
Recall: $sin30° = \frac{1}{2}$ and $cos30° = \frac{1}{2} \cdot \sqrt{3}$

$a = 3$



$(3)^2 + (3 \cdot \sqrt{3})^2$

$= 6^2$



$y = r \times sin\psi = 2a \times \frac{1}{2} = a$

$x = r * cos\psi$
$= 2a * \frac{1}{2}\sqrt{3}$
$= a \cdot \sqrt{3}$

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]

Interface used as a static type

Interface vs. Implementations

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

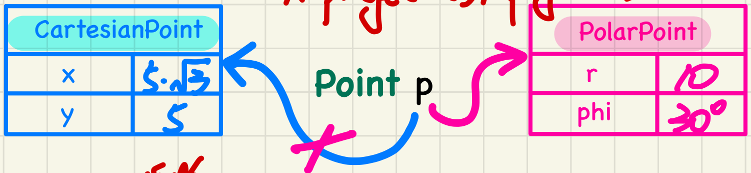
static method.

DT: CartesianPoint

DT: PolarPoint

Point p = new Point(); X not valid.

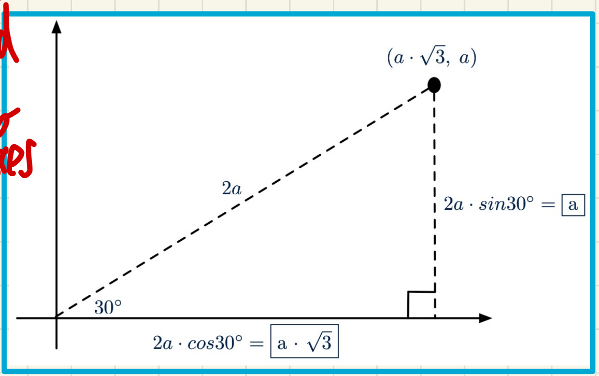
x p.getX() x p.getY()



CartesianPoint	
x	5·√3
y	5

Point p

PolarPoint	
r	10
phi	30°



$(a \cdot \sqrt{3},\ a)$
$2a$
$2a \cdot sin30° = \boxed{a}$
$30°$
$2a \cdot cos30° = \boxed{a \cdot \sqrt{3}}$

implementations deferred to sub classes

an abstract class where all methods are abstract

available across packages.

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

headers of methods

forced to impl.

implements

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

absolute position

```
public class PolarPoint implements Point {
  private double phi;
  private 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; }
}
```

relative position

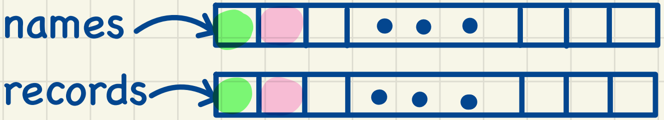
↳ measured in radians.

Lecture 7

Part A

Generics in Java - General Book: Storage vs. Retrieval

General Book



Supplier

```
public class Book {
  private String[] names;
  private Object[] records;
  /* add a name-record pair to the book */
  public void add (String name, Object record) { ... }
  /* return the record associated with a given name */
  public Object get (String name) { ... } }
```

STORAGE

arg object's ST must be a descendant of Object

RETRIEVAL

return value's DT must be a descendant of Object

Client

any objects can be added

```
Date birthday; String phoneNumber;
Book b; boolean isWednesday;
b = new Book();
phoneNumber = "416-67-1010";
b.add ("Suyeon", phoneNumber);
birthday = new Date(1975, 4, 10);
b.add ("Yuna", birthday);
isWednesday = b.get("Yuna").getDay() == 4;
```

X

Call by value valid ∵ String, Date is a descendant of object.

X available on the DT of RV, not ST.

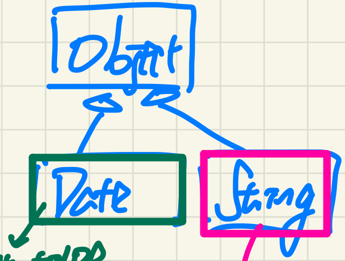
ST of RV is: Object

General Book: Retrieval from a Polymorphic Array

```
Date birthday; String phoneNumber;
Book b; boolean isWednesday;
b = new Book();
phoneNumber = "416-67-1010";
b.add ("Suyeon", phoneNumber);
birthday = new Date(1975, 4, 10);
b.add ("Yuna", birthday);
isWednesday = b.get("Yuna").getDay() == 4;
```

downward casting.

Client



cast type

DT: Date

DT of b.get("Suyeon")

anonymous cast with ST: Date

```
isWednesday = ((Date) b.get("Yuna")).getDay() == 4;
```

```
isWednesday = ((Date) b.get("Suyeon")).getDay() == 4;
```

↳ DT: String

Compile (downward cast) but ClassCast Excep.

object expression

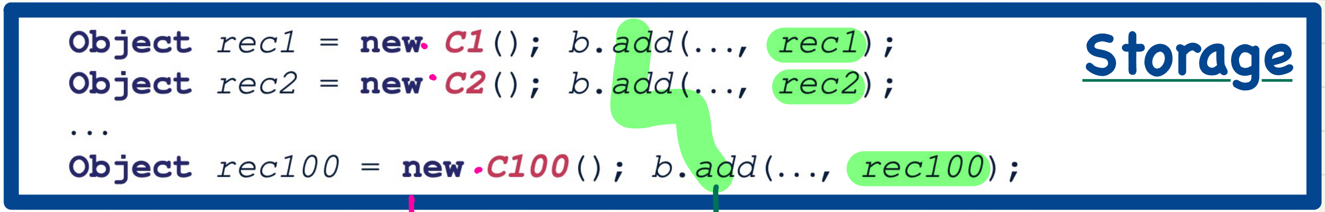
```
if (b.get("Suyeon") instanceof Date) {
  isWednesday = ((Date) b.get("Suyeon")).getDay() == 4;
}
```

evaluates to false

DT: String

↳ for any retrieval from a general book, it's required to have instanceof checks & type casts.

General Book violates Single Choice Principle

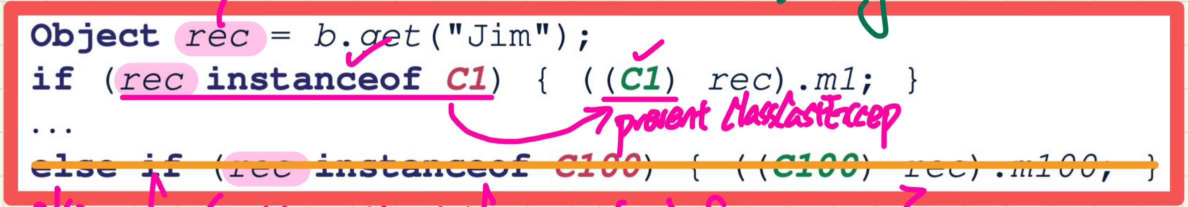


```
Object rec1 = new C1(); b.add(..., rec1);
Object rec2 = new C2(); b.add(..., rec2);
...
Object rec100 = new C100(); b.add(..., rec100);
```

Storage

storage

retrieval



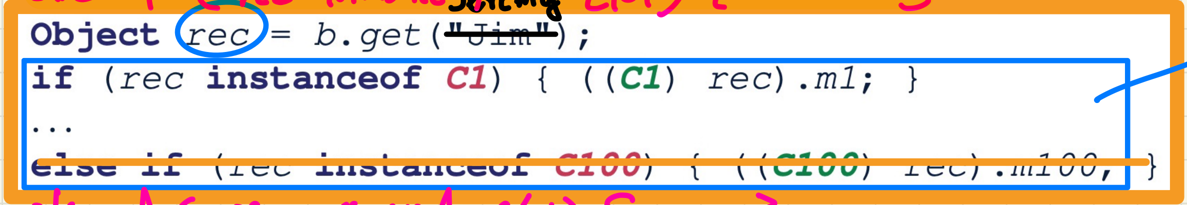
```
Object rec = b.get("Jim");
if (rec instanceof C1) { ((C1) rec).m1; }
...
~~else if (rec instanceof C100) { ((C100) rec).m100; }~~
```

prevent ClassCastExcep

else if (rec instanceof C101) { - - - }

"Jeremy"

Retrievals



```
Object rec = b.get(~~"Jim"~~);
if (rec instanceof C1) { ((C1) rec).m1; }
...
~~else if (rec instanceof C100) { ((C100) rec).m100; }~~
```

else if (rec instanceof C101) { - - - }

the same exhaustive checks on the DT of the retrieved record are repeated

What if a new type C101 is introduced?

What if type C100 becomes obselete?