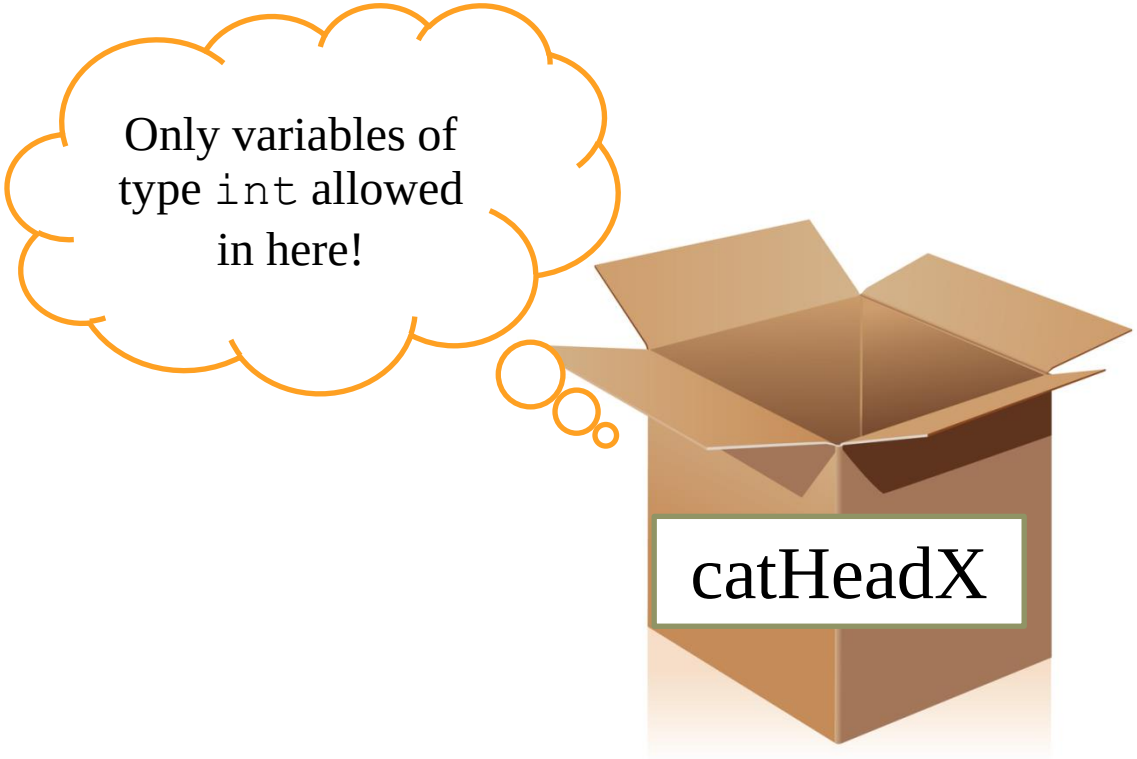
Polymorphism

Type-casting Objects

Polymorphism

Double Dispatching

Type-Casting Objects



Variable type → `int catHeadX;`

Type-casting primitives:

Changes the value to fit in a new type of box.

Type-casting objects:

Changes the reference type but not the object, affecting only what behaviors we have access to.

Type-casting Objects

Treats objects more generally, simplifying their use

Automatic type-casting occurs:

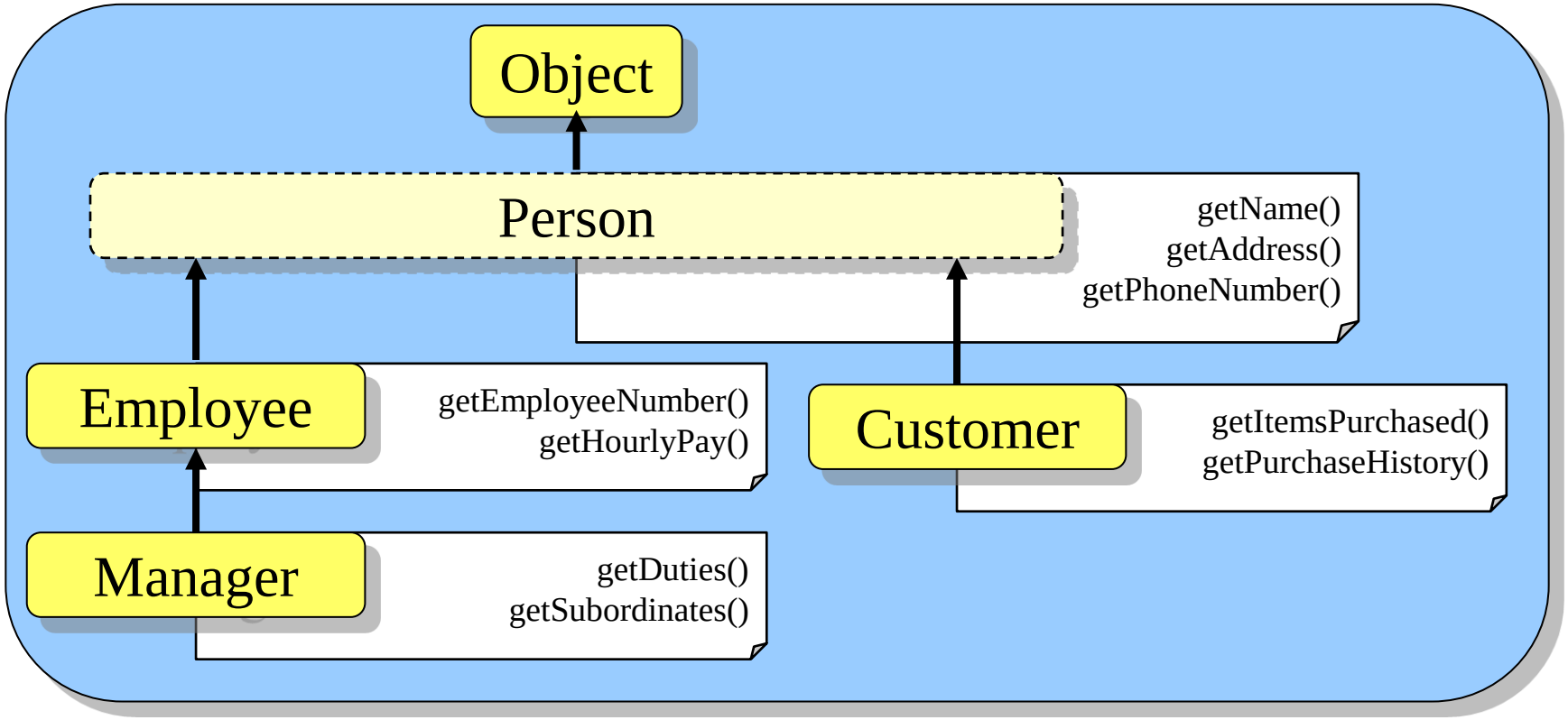
(1) When an object is assigned to a more general type

(2) When we pass in an object as a parameter with a more general type

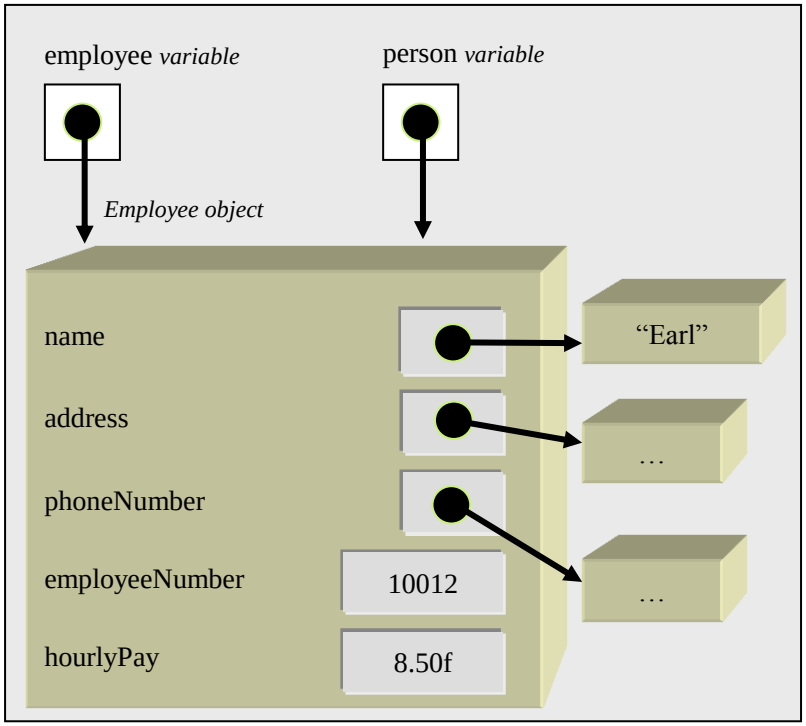
```
p = (Person)anEmployee;
c = (Customer)anArray[i];
b = (SavingsAccount)aBankAccount;
```

`p` is still an `Employee`, but will be treated as a `Person` now

```
p = (Person)anEmployee;
c = (Customer)anArray[i];
b = (SavingsAccount)aBankAccount;
```

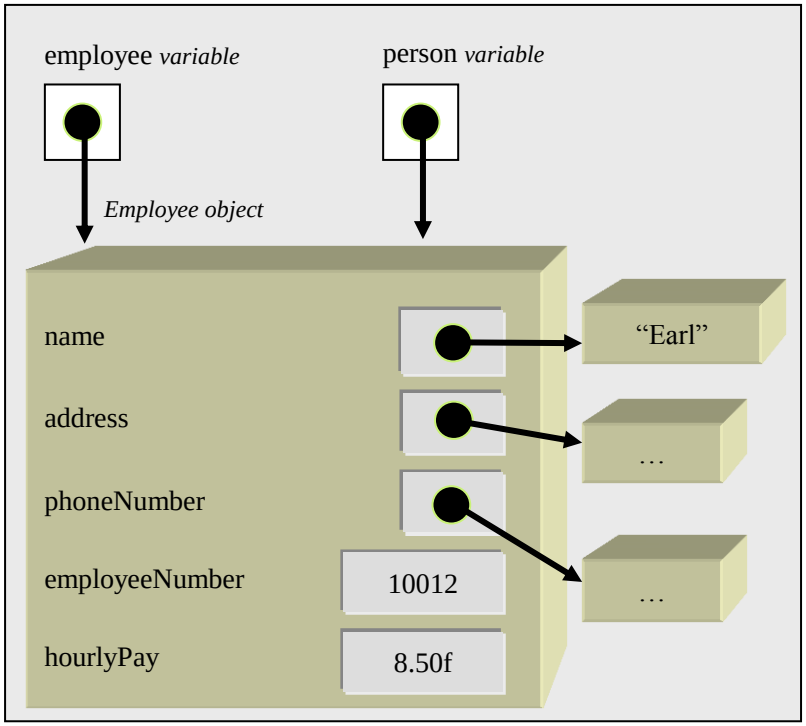

Object
Person
getName()
getAddress()
getPhoneNumber()
Employee
getEmployeeNumber()
getHourlyPay()
Customer
getItemsPurchased()
getPurchaseHistory()
Manager
getDuties()
getSubordinates()

```
Person person;
Employee employee;

employee = new Employee("Earl");
employee.getName();
employee.getAddress();
employee.getPhoneNumber();
employee.getEmployeeNumber();
employee.getHourlyPay();

person = (Person)employee;
person.getName();
person.getAddress();
person.getPhoneNumber();

person.getEmployeeNumber();
person.getHourlyPay();

((Employee)person).getEmployeeNumber();
((Employee)person).getHourlyPay();
```



```
Person person;
Employee employee;

employee = new Employee("Earl");
employee.getName();
employee.getAddress();
employee.getPhoneNumber();
employee.getEmployeeNumber();
employee.getHourlyPay();

person
person
person
person
```

Calling `Employee` methods, so this is all ok

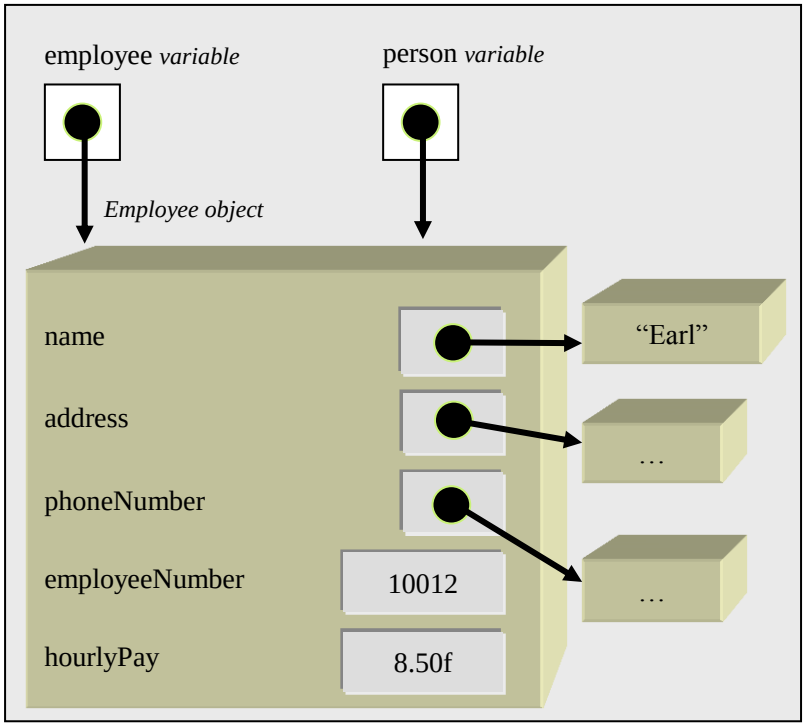



```
person.getEmployeeNumber();
person.getHourlyPay();

((Employee)person).getEmployeeNumber();
((Employee)person).getHourlyPay();
```

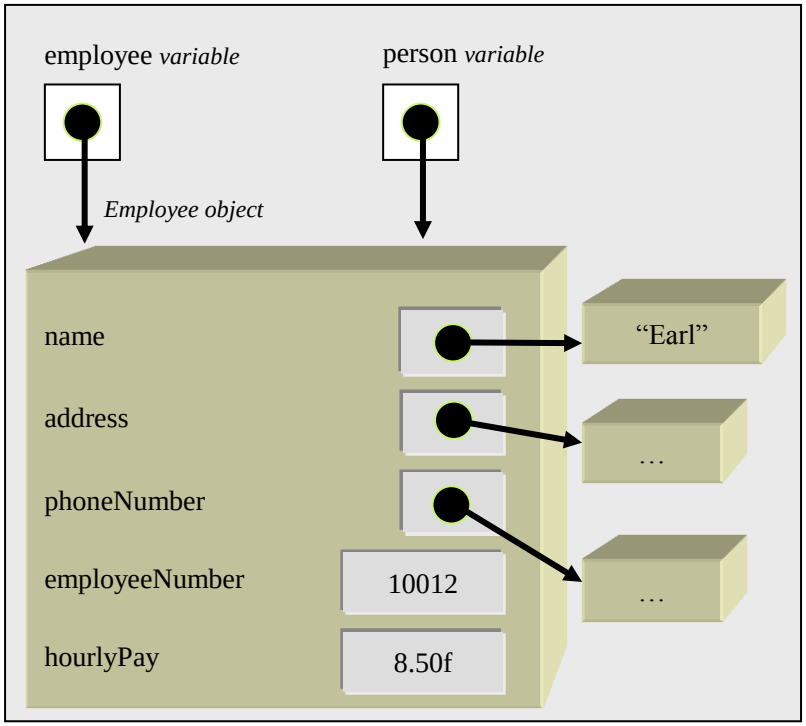
```
Person person;
Employee employee;

employee = new Employee("Earl");
employee.getName();
er
er
er
er
er
```

Treat Earl as a `Person` instead of an `Employee`

```
person = (Person)employee;
person.getName();
person.getAddress();
person.getPhoneNumber();

person.getEmployeeNumber();
person.getHourlyPay();

((Employee)person).getEmployeeNumber();
((Employee)person).getHourlyPay();
```

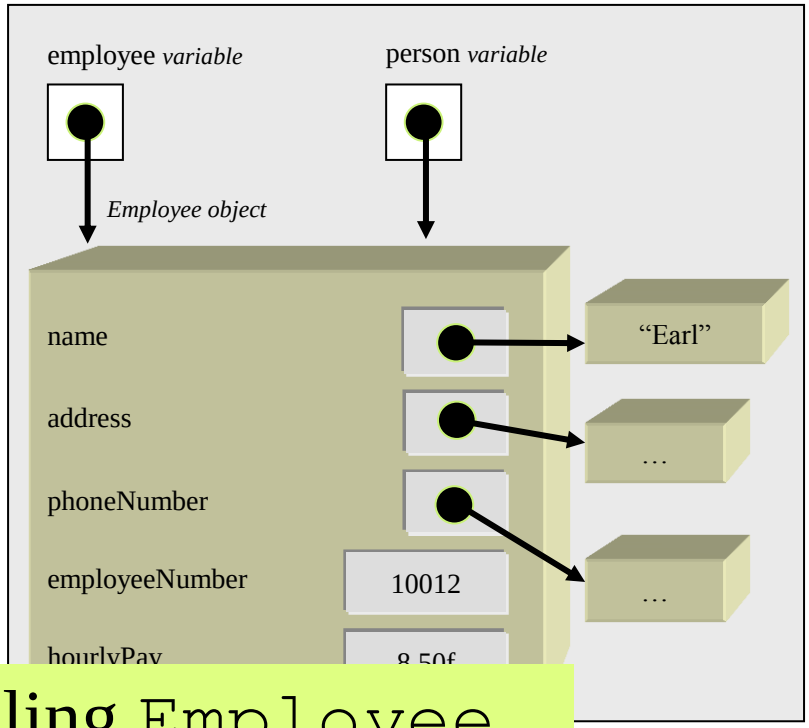


```
Person person;
Employee employee;

employee = new Employee("Earl");
employee.getName();
employee.getAddress();
employee.getPhoneNumber();
employee.getEmployeeNumber();
employee.getHourlyPay();

person = (Person)employee;
person.getName();
person.getAddress();
person.getPhoneNumber();

person.ge
person.ge

((Employe                    r();
((Employe
```



Calling `Person` methods, so this is all ok

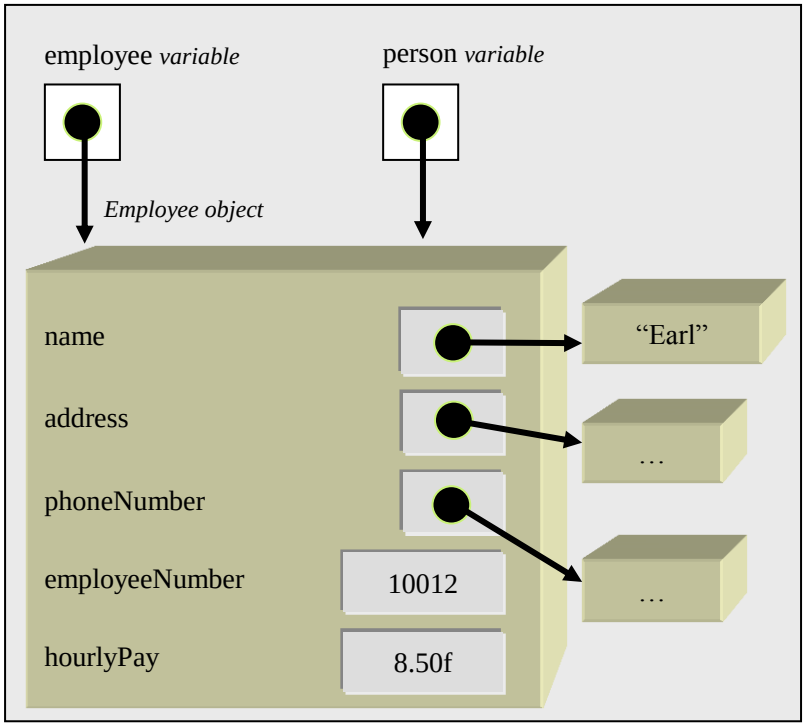
```
Person person;
Employee employee;

employee = new Employee("Earl");
employee.getName();
employee.getAddress();
employee.getPhoneNumber();
employee.getEmployeeNumber();
employee.getHourlyPay();

person = (Person)employee;
person.getName();
person.getAddress();
person.getPhoneNumber();

person.getEmployeeNumber();
person.getHourlyPay();
```



Calling `Employee` methods – will not compile

```
((Employee)person).getEmployeeNumber();
((Employee)person).getHourlyPay();
```

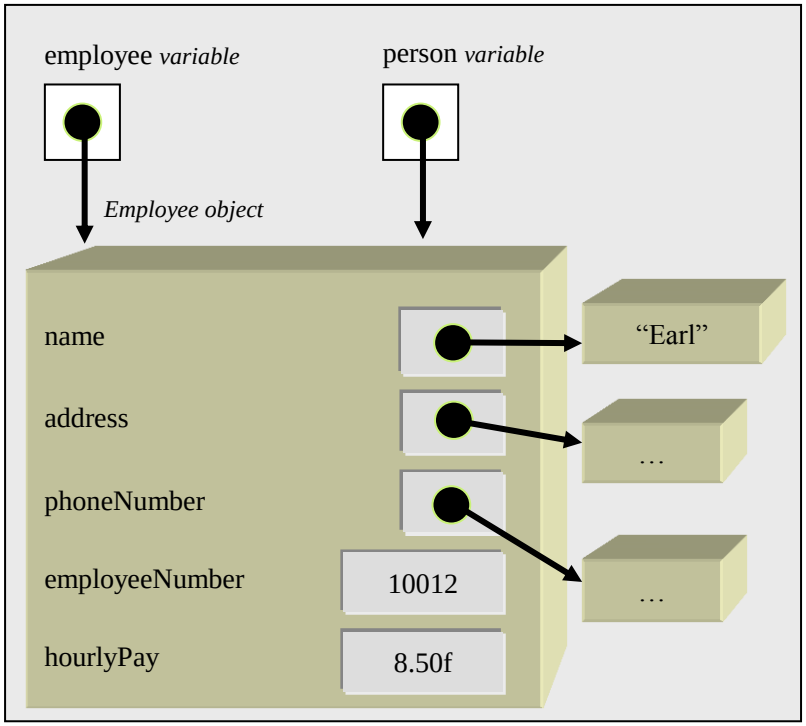
```
Person person;
Employee employee;

employee = new Employee("Earl");
employee.getName();
employee.getAddress();
employee.getPhoneNumber();
employee.getEmployeeNumber();
employee.getHourlyPay();

person = (Person)employee;
person.getName();
person.getAddress();
```



Casting back to an `Employee` (works since it really is one)

```
((Employee)person).getEmployeeNumber();
((Employee)person).getHourlyPay();
```

```
Person person;
Employee employee;

employee = new Employee("Earl");
employee.getName();
employee.getAddress();
employee.getPhoneNumber();
employee.getEmployeeNumber();
employee.getHourlyPay();

person = (Person)employee;
person.getName();
person.getAddress();
person.getPhoneNumber();

person.getEmployeeNumbe
person.getHourlyPay();
```



Calling `Employee` methods works now

```
((Employee)person).getEmployeeNumber();
((Employee)person).getHourlyPay();
```

Poll Everywhere Question

What will the result be of compiling and running the following code?

```
public class Fruit { … }
public class Apple extends Fruit { … }
public class MacintoshApple extends Apple { … }

public class Tester
{
   public static void main(String[] args)
   {
      Fruit fruit;
      Apple apple;
      MacintoshApple macintosh;

      fruit = new Fruit();
      apple = (Apple)fruit;
      macintosh = new MacintoshApple();
      System.out.println((Apple)macintosh);
   }
}
```

Text 37607

112150
`MacintoshApple`'s `toString()` used to print `macintosh`

112202
`Apple`'s `toString()` used to print `macintosh`

112204
compile error

112208
run-time error

Polymorphism

Polymorphism

the ability to use the same behavior for objects of different types

Polymorphism

"the ability (in programming) to present the same interface for differing underlying forms (data types)"

http://stackoverflow.com/questions/1031273/what-is-polymorphism-what-is-it-for-and-how-is-it-used

Double Dispatching

```
public class Shape
{
    ...
    public void draw() { ... }
}


public class Circle extends Shape
{
    ...
    public void draw() { ... }
}

public class Triangle extends Shape
{
    ...
    public void draw() { ... }
}

public class Rectangle extends Shape
{
    ...
    public void draw() { ... }
}
```

```
Circle c = new Circle(20);
Triangle t = new Triangle(10, 20, 30);
Rectangle r = new Rectangle(10, 10, 20, 20);

ArrayList<Shape> shapes =  new ArrayList<Shape>();
shapes.add(c);
shapes.add(t);
shapes.add(r);
```

```
Circle c = new Circle(20);
Triangle t = new Triangle(10, 20, 30);
Rectangle r = new Rectangle(10, 10, 20, 20);

ArrayList<Shape> shapes =  new ArrayList<Shape>();
shapes.add(c);
shapes.add(t);
shapes.add(r);
```

All are stored as `Shape` references (`c`, `t`, and `r` are implicitly cast)

Given any `Shape s`, how can we use the right `draw()` method?

```
Shape s = shapes.get(n);
```

Given any `Shape s`, how can we use the right `draw()` method?

```
if (s instanceof Circle)
    ((Circle)s).draw();

if (s instanceof Triangle)
    ((Triangle)s).draw();

if (s instanceof Rectangle)
    ((Rectangle)s).draw();
```

Given any `Shape s`, how can we use the right `draw()` method?

```
if (s instanceof Circle)
    ((Circle)s).draw();

if (s instanceof Triangle)
    ((Triangle)s).draw();

if (s instanceof Rectangle)
    ((Rectangle)s).draw();
```

Make use of polymorphism – Java knows what object is *really* in `s`

```
Shape s = shapes.get(n);
s.draw();
```

What if we had a pen class that can draw many different shapes?

What if we had a pen class that can draw many different shapes?

```
Pen  aPen = new Pen();

aPen.draw(aCircle);
aPen.draw(aTriangle);
aPen.draw(aRectangle);
```

```
public class Pen
{
    ...
    public void draw(Circle aCircle)
    {
        // code that draws a Circle
    }
    public void draw(Triangle aTriangle)
    {
        // code that draws a Triangle
    }
    public void draw(Rectangle aRectangle)
    {
        // code that draws a Rectangle
    }
}
```

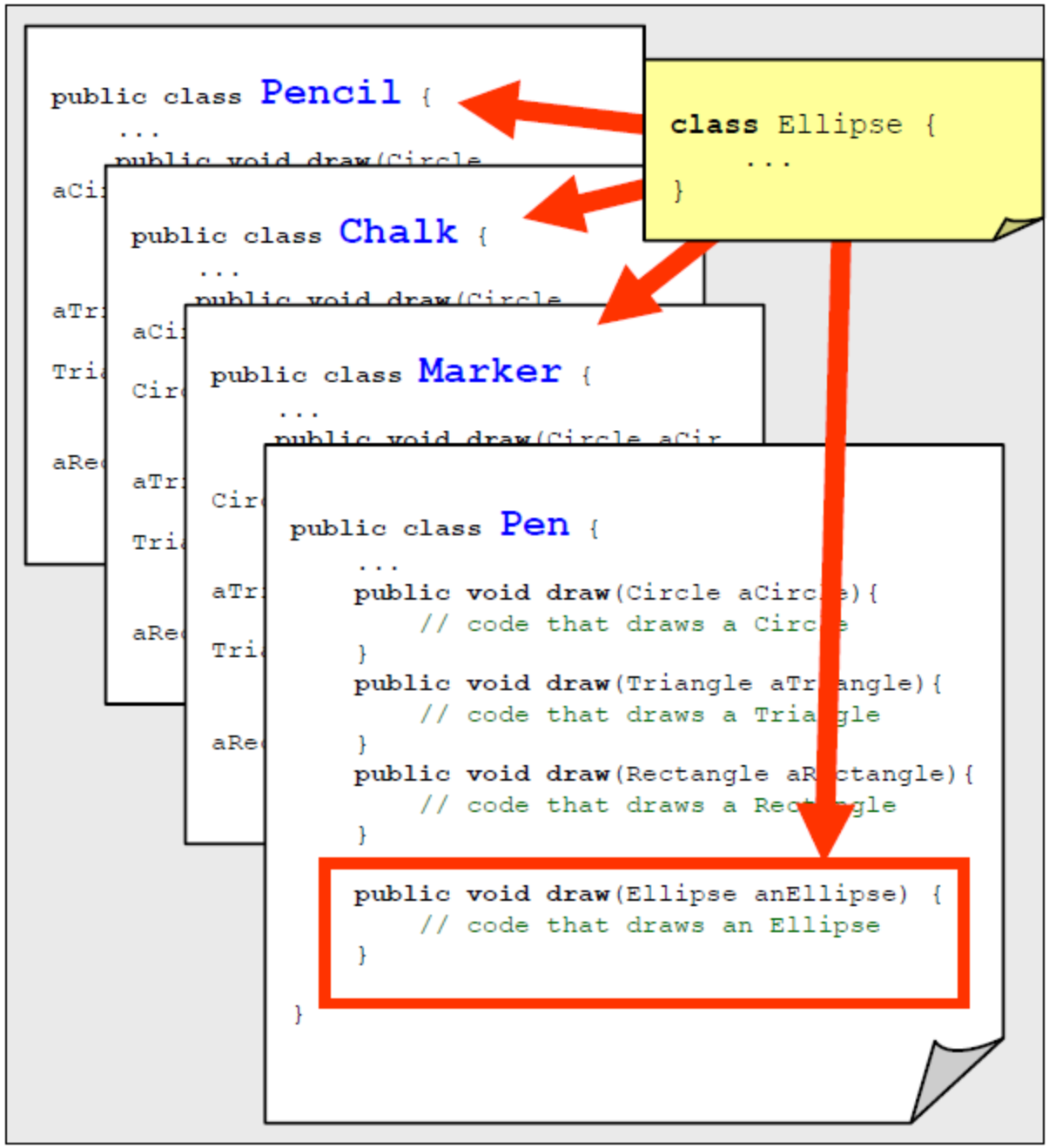
What if we also wanted to have a `Pencil`, `Chalk`, and `Marker` class now?

```
public class Pen
{
    ...
    public void draw(Circle aCircle)
    {
        // code that draws a Circle
    }
    public void draw(Triangle aTriangle)
    {
        // code that draws a Triangle
    }
    public void draw(Rectangle aRectangle)
    {
        // code that draws a Rectangle
    }
}
```

What if we also wanted to have a `Pencil`, `Chalk`, and `Marker` class now?

```
public class Pen
{
    ...
    public void draw(Circle aCircle)
    {
        // code that draws a Circle
    }
    public void draw(Triangle aTriangle)
    {
        // code that draws a Triangle
    }
    public void draw(Rectangle aRectangle)
    {
        // code that draws a Rectangle
    }
}
```

What if we wanted to add shapes?


public class Pencil {
...
public void draw(Circle
public class Chalk {
...
public void draw(Circle
public class Marker {
...
public void draw(Circle aCir
public class Pen {
...
public void draw(Circle aCircle){
// code that draws a Circle
}
public void draw(Triangle aTriangle){
// code that draws a Triangle
}
public void draw(Rectangle aRectangle){
// code that draws a Rectangle
}
public void draw(Ellipse anEllipse) {
// code that draws an Ellipse
}
}
class Ellipse {
...
}

Solution:

Double-dispatching
(Pass the buck, calling another method to do the work)

```
public class Pen
{
    ...
    public void draw(Shape anyShape)
    {
        if (anyShape instanceof Circle)
            // Do the drawing for circles
        if (anyShape instanceof Triangle)
            // Do the drawing for triangles
        if (anyShape instanceof Rectangle)
            // Do the drawing for rectangles}
    }
}
```

```
public class Pen
{
    ...
    public void draw(Shape anyShape)
    {
        if (anyShape instanceof Circle)
            // Do the drawing for circles
        if (anyShape instanceof Triangle)
            // Do the drawing for triangles
        if (anyShape instanceof Rectangle)
            // Do the drawing for rectangles}
    }
}
```

Avoid this by making the shape draw itself with the pen

```
public class Circle extends Shape
{
    ...
    public void drawWith(Pen aPen) { ... }
}


public class Triangle extends Shape
{
    ...
    public void drawWith(Pen aPen) { ... }
}


public class Rectangle extends Shape
{
    ...
    public void drawWith(Pen aPen) { ... }
}
```

```
public class Circle extends Shape
{
    public void drawWith(Pen aPen) { ... }
}
```

Who knows better how to draw a shape than the shape itself?

```
public class T
{
    ...
    public void drawWith(Pen aPen) { ... }
}
```

```
public class Rectangle extends Shape
{
    ...
    public void drawWith(Pen aPen) { ... }
}
```

```
public class Pen
{
    ...
    public void draw(Shape aShape)
    {
        aShape.drawWith(this);
    }
}
```

```
public class Pen
{
    ...
    public void draw(Shape aShape)
    {
        aShape.drawWith(this);
    }
}
```

Double dispatch! Draw your own self, `Shape`!