ECE 462
Object-Oriented Programming using C++ and Java

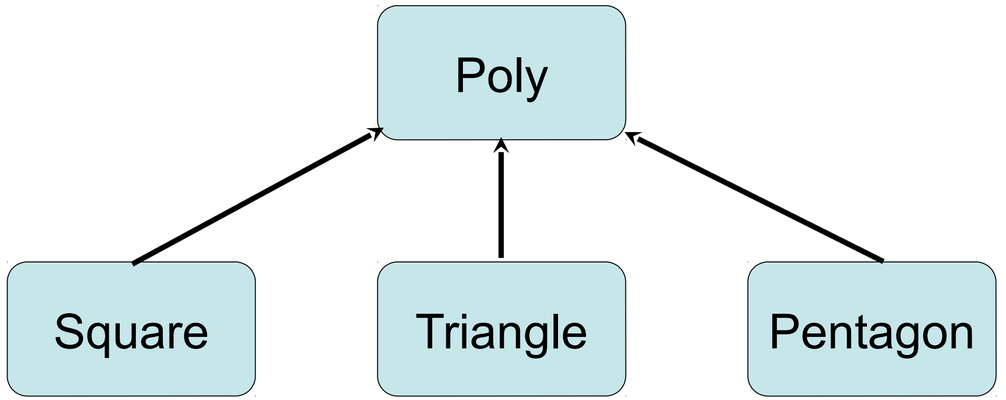
Inheritance and Polymorphism

A little terminology - methods and functions

- Methods are any function that is declared within a class and can access class information
- All functions in Java are methods. As we will see later this is not true for C++.

Override Behavior

- Poly can also support getArea.
- Derived classes (such as Triangle, Square, and Pentagon) **may** have better (faster) ways to getArea than Polygon.
- getArea is implemented in Poly and optionally in its derived classes.
- A Poly object calls getArea in Poly
- A Square object calls getArea in Square ***if*** getArea ***is*** implemented in Square.
- But, a Pentagon object calls getArea in Poly ***if*** getArea ***is not*** implemented in Pentagon.



Let’s look at a Poly, etc., class in Java

```
// get access to math routines
import java.lang.Math;

public class Poly {
        private int n; // number of sides
        private double s; // length of side

        public Poly(int fn, double fs) {
                n = fn;
                s = fs;
        }

        public String toString( ) {
                return n+" "+s;
        }

        public double getLenSides( ) {
           return s;
        }

        public double getArea( ) {
           System.out.println(" poly area");
           return (s*s*n)/(4*Math.tan(Math.PI/n));
        }
}
```

Poly.java

```
// get access to math routines
import java.lang.Math;

public class Poly {
        private int n; // number of sides
        private double s; // length of side

        public Poly(int fn, double fs) {
                n = fn;
                s = fs;
        }

        public String toString( ) {
                return n+" "+s;
        }

        public double getLenSides( ) {
           return s;
        }

        public double getArea( ) {
           System.out.println(" poly area");
           return (s*s*n)/(4*Math.tan(Math.PI/n));
        }
}
```

Poly.java

```
// get access to math routines
import java.lang.Math;

public class Poly {
        private int n; // number of sides
        private double s; // length of side

        public Poly(int fn, double fs) {
                n = fn;
                s = fs;
        }

        public String toString( ) {
                return n+" "+s;
        }

        public double getLenSides( ) {
           return s;
        }

        public double getArea( ) {
           System.out.println(" poly area");
           return (s*s*n)/(4*Math.tan(Math.PI/n));
        }
}
```

private int n;
private double s;
Poly(int, double)
ToString( )
getArea( );

Poly.java

```
// get access to math routines
import java.lang.Math;

public class Poly {
        private int n; // number of sides
        private double s; // length of side

        public Poly(int fn, double fs) {
                n = fn;
                s = fs;
        }

        public String toString( ) {
                return n+" "+s;
        }

        public double getLenSides( ) {
           return s;
        }

        public double getArea( ) {
           System.out.println(" poly area");
           return (s*s*n)/(4*Math.tan(Math.PI/n));
        }
 }
```

Red declares variables to hold the state of a Poly object.
Green defines a *constructor* of a poly object. Used to define the initial state of the object when it is formed.
Blue are *methods* that defines the actions of a Poly object

Poly.java

```
// get access to math routines
import java.lang.Math;

public class Poly {
        private int n; // number of sides
        private double s; // length of side

        public Poly(int fn, double fs) {
                n = fn;
                s = fs;
        }

        public String toString( ) {
                return n+" "+s;
        }

        public double getLenSides( ) {
           return s;
        }

        public double getArea( ) {
           System.out.println(" poly area");
           return (s*s*n)/(4*Math.tan(Math.PI/n));
        }
}
```

Poly.java

```
// get access to math routines
import java.lang.Math;

public class Poly {
        private int n; // number of sides
        private double s; // length of side

        public Poly(int fn, double fs) {
                n = fn;
                s = fs;
        }

        public String toString( ) {
                return n+" "+s;
        }

        public double getLenSides( ) {
           return s;
        }

        public double getArea( ) {
           System.out.println(" poly area");
           return (s*s*n)/(4*Math.tan(Math.PI/n));
        }
}
```

Poly.java

```
// get access to math routines
import java.lang.Math;

public class Poly {
        private int n; // number of sides
        private double s; // length of side

        public Poly(int fn, double fs) {
                n = fn;
                s = fs;
        }

        public String toString( ) {
                return n+" "+s;
        }

        public double getLenSides( ) {
           return s;
        }

        public double getArea( ) {
           System.out.println(" poly area");
           return (s*s*n)/(4*Math.tan(Math.PI/n));
        }
}
```

Square.java

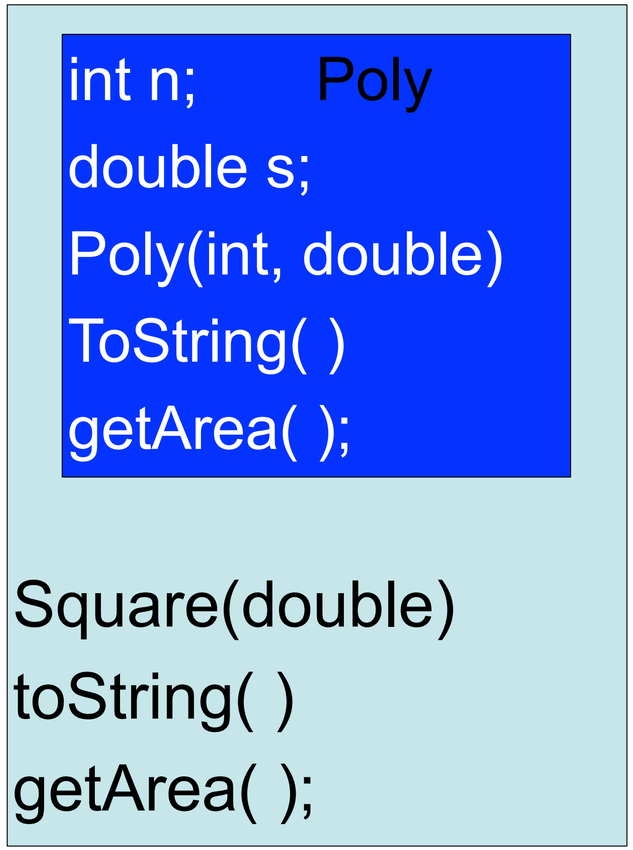
```
public class Square extends Poly {

        public Square(double fs) {
                super(4,fs);
        }

        public String toString( ) {
                return "4 "+getLenSides( );
        }

        public double getArea( ) {
           System.out.println(" square area");
           return getLenSides( )*getLenSides( );
        }
}
```

Square.java

```
public class Square extends Poly {

        public Square(double fs) {
                super(4,fs);
        }

        public String toString( ) {
                return "4 "+getLenSides( );
        }

        public double getArea( ) {
           System.out.println(" square area");
           return getLenSides( )*getLenSides( );
        }
}
```

Square.java

```
public class Square extends Poly {

        public Square(double fs) {
                super(4,fs);
        }

        public String toString( ) {
                return "4 "+getLenSides( );
        }

        public double getArea( ) {
           System.out.println(" square area");
           return getLenSides( )*getLenSides( );
        }
}
```



Square.java

```
public class Square extends Poly {

        public Square(double fs) {
                super(4,fs); // what if no base class constructor called here?
                             // when is base class constructor called?
        }

        public String toString( ) {
                return "4 "+getLenSides( );
        }

        public double getArea( ) {
           System.out.println(" square area");
           return getLenSides( )*getLenSides( );
        }
```

Square.java

```
public class Square extends Poly {

        public Square(double fs) {
                super(4,fs);
        }

        public String toString( ) {
                return "4 "+getLenSides( );
        }

        public double getArea( ) {
           System.out.println(" square area");
           return getLenSides( )*getLenSides( );
        }
}
```

Square.java

```
public class Square extends Poly {

        public Square(double fs) {
                super(4,fs);
        }

        public String toString( ) {
                return "4 "+getLenSides( );
        }

        public double getArea( ) {
           System.out.println(" square area");
           return getLenSides( )*getLenSides( );
        }
}
```

Pentagon.java

```
public class Pentagon extends Poly {

        public Pentagon(double fs) {
                super(5, fs);
        }

        public String toString( ) {
                return "Pentagon with side of length"+getLenSides( );
        }
}
```

Pentagon.java

```
public class Pentagon extends Poly {

        public Pentagon(double fs) {
                super(5,fs);
        }

        public String toString( ) {
                return "Pentagon with side of length"+getLenSides( );
        }
}
```

int n; Poly
double s;
Poly(int, double)
ToString( )
getArea( );

Pentagon(double)
toString( )

Pentagon.java

```
public class Pentagon extends Poly {

        public Pentagon(double fs) {
                super(5,fs);
        }

        public String toString( ) {
                return "Pentagon with side of length"+getLenSides( );
        }
}
```

Test.java -- contains the *main* function

```
public class Test {

  public static void main(String[] args) {

     Poly p1 = new Poly(6, 2.0);
     Square s1 = new Square(2.0);
     Pentagon pe1 = new Pentagon(2.0);

     System.out.println("Poly p1 is "+p1+", area is "+p1.getArea( ));
     System.out.println(" ");

     System.out.println("Square s1 is "+s1+", area is "+s1.getArea( ));
     System.out.println(" ");

     System.out.println("Pentagon pe1 is "+pe1+", area is "+pe1.getArea( ));
     System.out.println(" ");

     p1 = s1;
     System.out.println("Poly p1 is "+p1+", area is "+p1.getArea( ));
  }
}
```

Test.java -- contains the *main* function

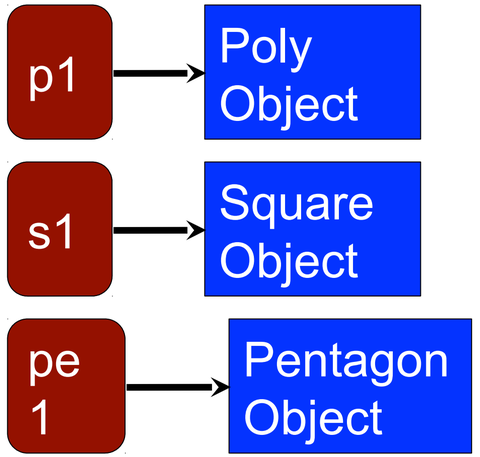
```
public class Test {

  public static void main(String[] args) {

     Poly p1 = new Poly(6, 2.0);
     Square s1 = new Square(2.0);
     Pentagon pe1 = new Pentagon(2.0);

     System.out.println("Poly p1 is "+p1+", area is "+p1.getArea( ));
     System.out.println(" ");

     System.out.println("Square s1 is "+s1+", area is "+s1.getArea( ));
     System.out.println(" ");

     System.out.println("Pentagon pe1 is "+pe1+", area is "+pe1.getArea( ));
     System.out.println(" ");

     p1 = s1;
     System.out.println("Poly p1 is "+p1+", area is "+p1.getArea( ));
   }
}
```

Test.java -- contains the *main* function

```
public class Test {

  public static void main(String[] args) {

     Poly p1 = new Poly(6, 2.0);
     Square s1 = new Square(2.0);
     Pentagon pe1 = new Pentagon(2.0);

     System.out.println("Poly p1 is "+p1+", area is "+p1.getArea( ));
     System.out.println(" ");

     System.out.println("Square s1 is "+s1+", area is "+s1.getArea( ));
     System.out.println(" ");

     System.out.println("Pentagon pe1 is "+pe1+", area is "+pe1.getArea( ));
     System.out.println(" ");

     p1 = s1;
     System.out.println("Poly p1 is "+p1+", area is "+p1.getArea( ));
   }
}
```

Test.java -- contains the *main* function

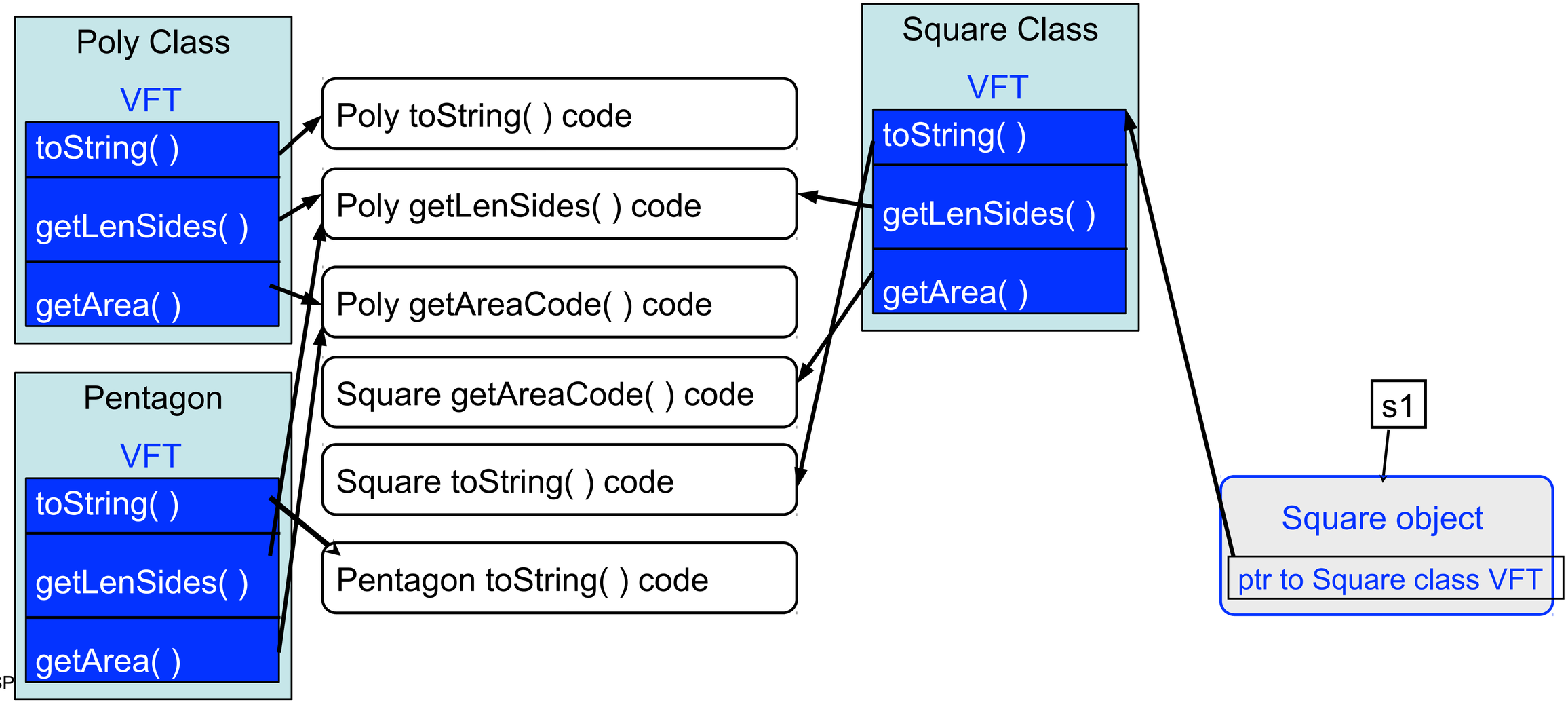
```
public class Test {

  public static void main(String[] args) {

     Poly p1 = new Poly(6, 2.0);
     Square s1 = new Square(2.0);
     Pentagon pe1 = new Pentagon(2.0);

     System.out.println("Poly p1 is "+p1+", area is "+p1.getArea( ));
     System.out.println(" ");

     System.out.println("Square s1 is "+s1+", area is "+s1.getArea( ));
     System.out.println(" ");

     System.out.println("Pentagon pe1 is "+pe1+", area is "+pe1.getArea( ));
     System.out.println(" ");

     p1 = s1;
     System.out.println("Poly p1 is "+p1+", area is "+p1.getArea( ));
  }
}
```



Test.java

```
public class Test {

  public static void main(String[] args) {
. . .

      System.out.println("Poly p1 is "+p1+", area is "+p1.getArea( ));
      System.out.println(" ");
poly area
Poly p1 is 6 2.0, area is 10.392304845413264
      System.out.println("Square s1 is "+s1+", area is "+s1.getArea( ));
      System.out.println(" ");
square area
Square s1 is 4 2.0, area is 4.0
      System.out.println("Pentagon pe1 is "+pe1+", area is "+pe1.getArea( ));
      System.out.println(" ");
 poly area
Pentagon pe1 is Pentagon with side of length 2.0, area is 6.881909602355868
      p1 = s1;
      System.out.println("Poly p1 is "+p1+", area is "+p1.getArea( ));
???????? area
Poly p1 is 4 2.0, area is 4.0
```

How is it known which *getArea* to call?
Virtual Function Tables



Another Polymorphism example

```
import java.io.*;

public class Foo {

   private final String fooString;

   public Foo( ) {fooString = null;}
   public Foo(String ln) {fooString = ln;}
   public void print( ) {System.out.println("Foo: "+fooString);}
}
```

```
import java.io.*;

public class DFoo extends Foo {

   private final String dfooString;

   public DFoo(String ln) {dfooString = ln;}
   public void print( ) {System.out.println("DFoo: "+dfooString);}
}
```

```
import java.io.*;

class Test {

   public static void main(String args[ ]) {
      Foo f = new Foo("a new foo");
      f.print( );

      DFoo d = new DFoo("a new dfoo");
      d.print( );

      ((Foo) d).print( );

      f = d;
      f.print( );
   }
}
```

From java/baseDerived/

```
import java.io.*;

class Test {

   public static void main(String args[]) {
      Foo f = new Foo("Foo object");
      f.print( );

      DFoo d = new DFoo("DFoo object");
      d.print( );

      ((Foo) d).print( );

      f = d;
      f.print( );
   }
}
```

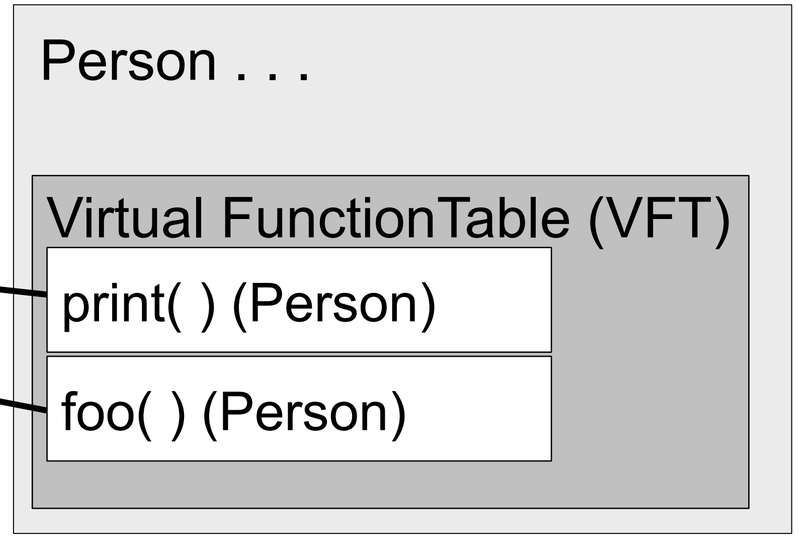
```
smidkiff% javac Test.java
smidkiff% java Test
Foo: Foo object
DFoo: DFoo object
DFoo: DFoo object
DFoo: DFoo object
[ece-76-55:code/java/baseDerived] smidkiff%
```

The class of the object on which the method is invoked is the class whose methods are called

Another Virtual Function Table (VFT) Example

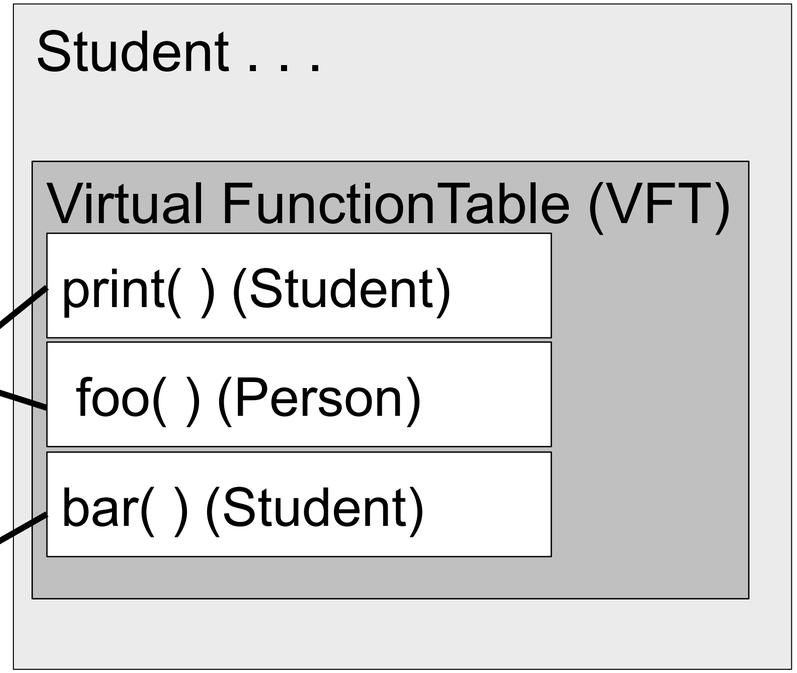
```
public class Person {
   final String p_lastName;
         final String p_firstName;

   public Person(String ln, String fn) {. . .}
   public void print( ) {. . .}
         public void foo( ) {. . .}
}
public class Student extends Person {
   String s_school;
   String s_major;

   public Student(. . .) {. . .}
         public void print( ) {. . .}
         public void bar( ) {. . .}
}
```

Person class VFT



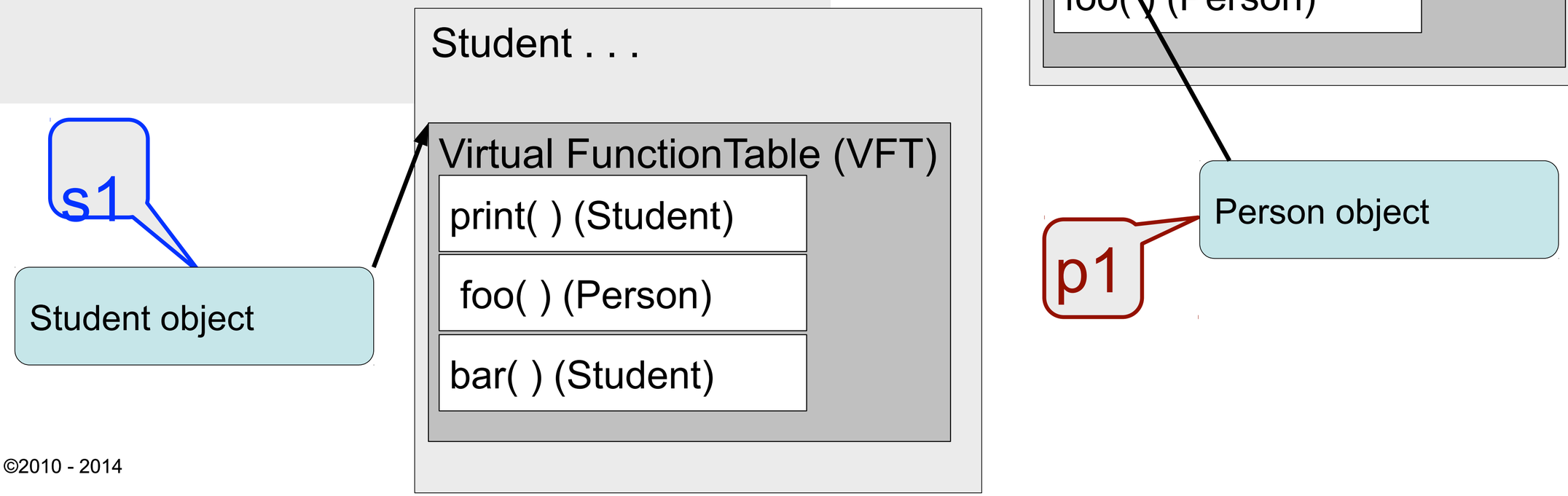
The Student class VFT

```
public class Person {
   final String p_lastName;
         final String p_firstName;

   public Person(String ln, String fn) {. . .}
   public void print( ) {. . .}
         public void foo( ) {. . .}
}
public class Student extends Person {
   String s_school;
   String s_major;

   public Student(. . .) {. . .}
         public void print( ) {. . .}
         public void bar( ) {. . .}
}
```



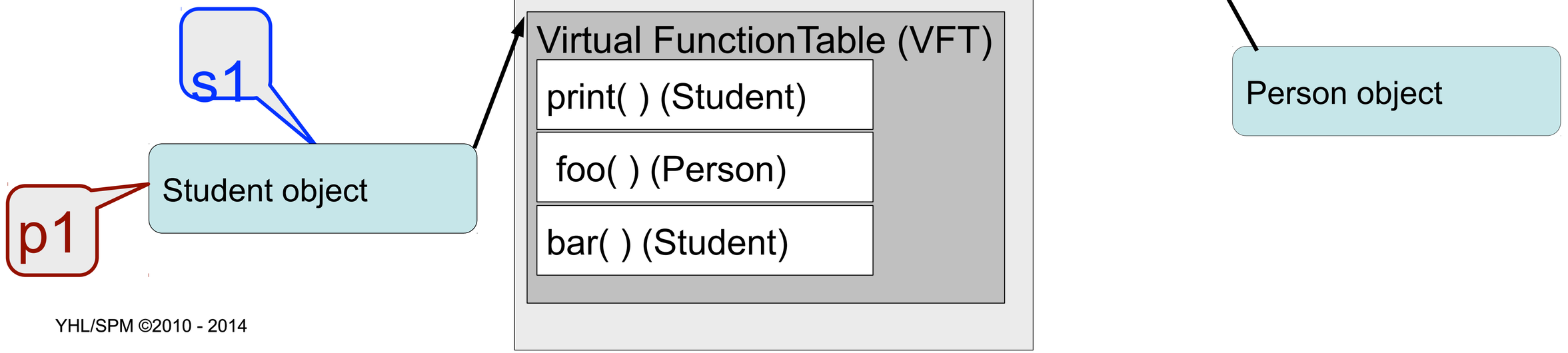
How the VFT enables polymorphic behavior

```
public static void main(. . .) {
   Person p1 = new Person("Johnson", "Tom");
   Student s1 = new Student("Smith", "Mary",
                                    "Purdue", "ECE");

   . . .
   p1 = s1;
   . . .
```



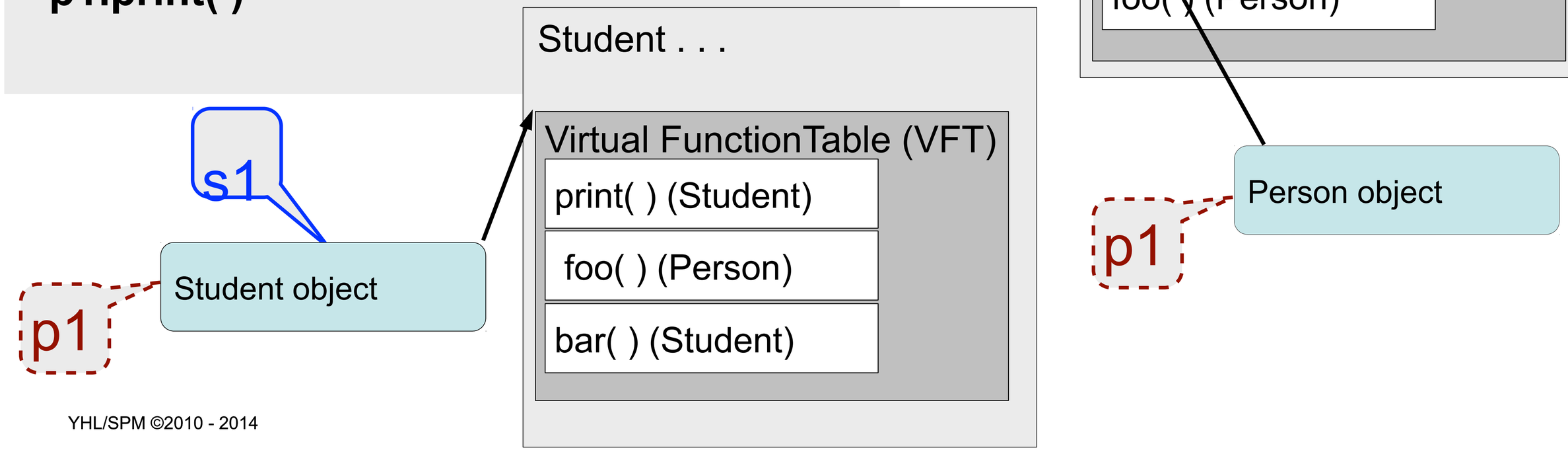
This will show polymorphic behavior

```
public static void main(. . .) {
   Person p1 = new Person("Johnson", "Tom");
   Student s1 = new Student("Smith", "Mary",
                                    "Purdue", "ECE");

   . . .
   p1 = s1;
   p1.print( )
```



What about this case? (new slide)

```
public static void main(. . .) {
   Person p1 = new Person("Johnson", "Tom");
   Student s1 = new Student("Smith", "Mary",
                                    "Purdue", "ECE");
   . . .
   if (some expression) p1 = s1;
   p1.print( )
```



Forcing base methods to be invoked in Java

```
public class Foo {
   private final String fooString;

   public Foo( ) {fooString = null;}
   public Foo(String ln) {fooString = ln;}
   public void print( ) {
      System.out.println("Foo: "+fooString);
   }
}
```

```
public class DFoo extends Foo {
   private final String dfooString;

   public DFoo(String ln) {dfooString = ln;}
   public void print( ) {
      System.out.print("DFoo, printing super: ");
      super.print( ); // invokes print in base
                      // (super) class
      System.out.println("DFoo: "+dfooString);
   }
}
```

From java/SuperInvoke/

A Java Gotcha

- *private* functions are not overridden - the *base print( )* will be called when using a reference to a base object (a *Foo* in this example).

```
public class Base {

   public Base( ) { }
   private void print( ) {
      System.out.println("Base print");
   }

   public void callPrint(Base b) {
      b.print( );
   }
}
```

```
public class Derived extends Base {

   public Derived( ) { }
   public void print( ) {
      // super.print( ); // invokes print in base
      System.out.print("Derived Print");
   }
}
public class Test {

   public static void main(String[ ] s) {
      Base b = new Base( );
      Derived  der = new Derived( );
      b.callPrint(der);
      der.callPrint(der);
   }
}
```

A Java Gotcha

- *private* functions are not overridden - the *base print( )* will be called when using a reference to a base object (a *Foo* in this example).

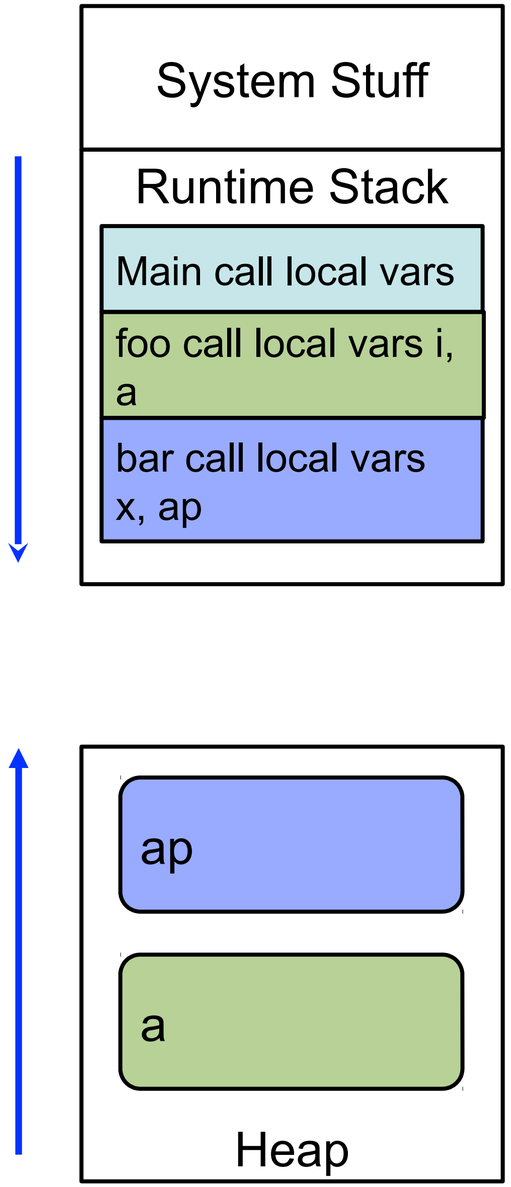
```
public class Base {

   public Base( ) { }
   private void print( ) {
      System.out.println("Base print");
   }

   public void callPrint(Base b) {
      b.print( );
   }
}
```

```
public class Derived extends Base {

   public Derived( ) { }
   public void print( ) {
      // super.print( ); // invokes print in base
      System.out.print("Derived Print");
   }
}
public class Test {

   public static void main(String[ ] s) {
      Base b = new Base( );
      Derived  der = new Derived( );
      b.callPrint(der);
      der.callPrint(der);
   }
}
```

```
Base print
Base print
```

Heaps and stacks and OO impurity in Java

- Java is not a *pure* OO language
- One source of impurity -- not everything in Java is an object
- Primitives, such as float, double, int, long, char, . . . are not objects
- String, arrays, instances of user defined classes are objects
- 0

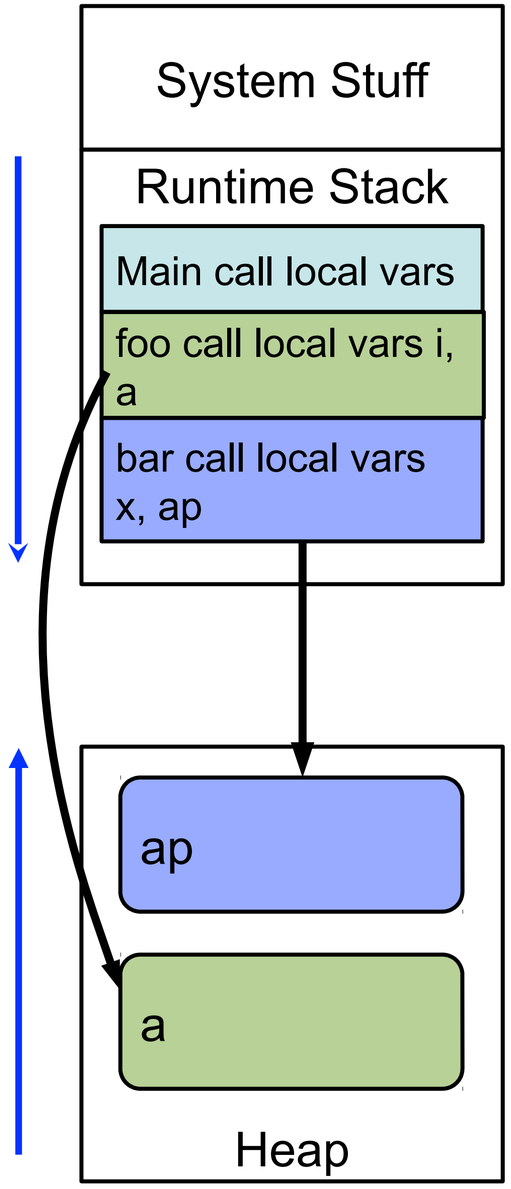



```
public class Beta inherits Alpha {

  public void bar(Alpha x) {
    Alpha ap = new Alpha( );
  }

  public void foo( ) {
    int i;
    Alpha a = new Alpha( );
    bar(a)
  }
```

- The variable *i* corresponds to storage that holds an *int*
- The variables *a* and *ap* correspond to storage that hold references to objects in the heap

Heaps and Java variables and objects

- Java is not a *pure* OO language
- One source of impurity -- no everything in Java is an object
- Primitives, such as float, double, int, long, char, . . . are not objects
- String, arrays, instances of user defined classes are objects

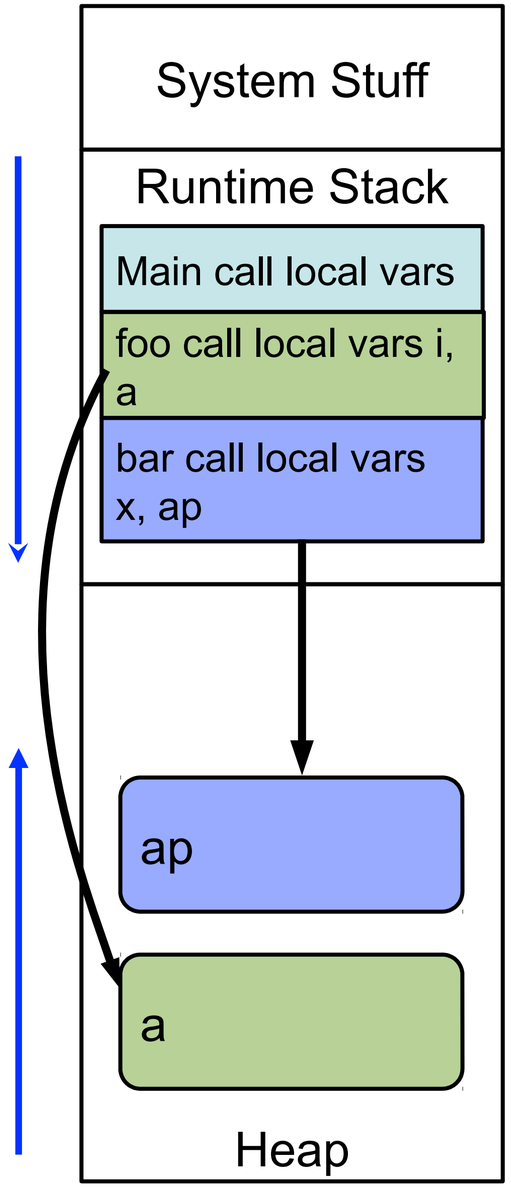



```
public class Beta extends Alpha {

  public void bar(Alpha x) {
    Alpha ap = new Alpha( );
  }

  public void foo( ) {
    int i;
    Alpha a = new Alpha( );
    a.bar(a)
  }
```

- The variable *i* corresponds to storage that holds an *int*
- The variables *a* and *ap* correspond to storage that hold references to objects in the heap

Heaps and Java variables and objects

- What happens when the heap runs into the stack either because of
  - stack growth
  - heap growth?
- If no space can be **garbage collected**, you'll get an *out of memory* condition.

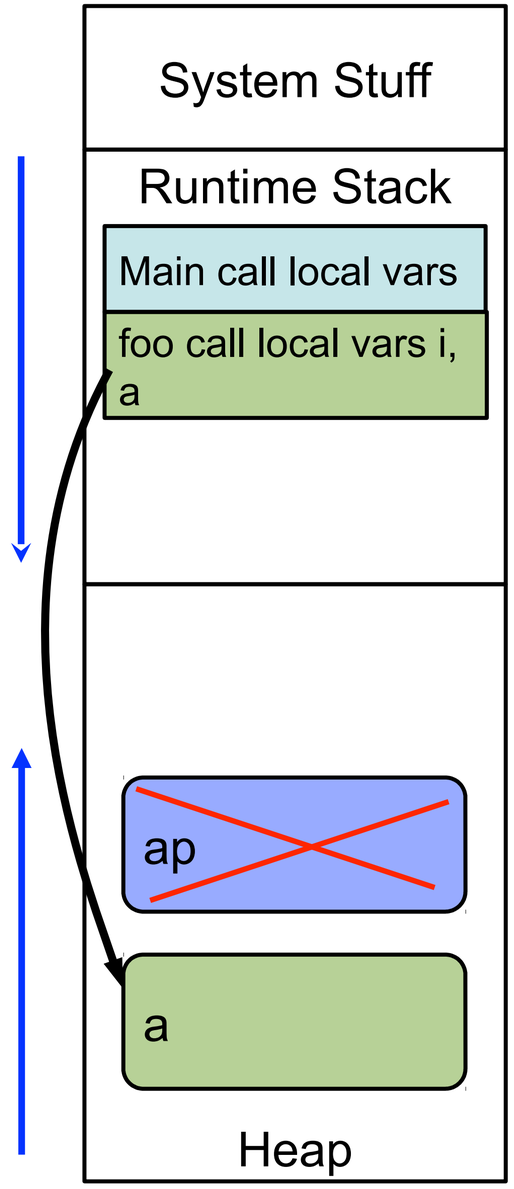



```
public class Beta inherits Alpha {

  public void bar(Alpha x) {
    Alpha ap = new Alpha( );
  }

  public void foo( ) {
    int i;
    Alpha a = new Alpha( );
    bar(a)
  }
```

- The variable *i* corresponds to storage that holds an *int*
- The variables *a* and *ap* correspond to storage that hold references to objects in the heap

Heaps and Java variables and objects

- What happens when the heap runs into the stack either because of
  - stack growth
  - heap growth?
- If no space can be **garbage collected**, you'll get an *out of memory* condition.




```
public class Beta inherits Alpha {

  public void bar(Alpha x) {
    Alpha ap = new Alpha( );
  }

  public void foo( ) {
    int i;
    Alpha a = new Alpha( );
    bar(a)
  }
```

- The variable *i* corresponds to storage that holds an *int*
- The variables *a* and *ap* correspond to storage that hold references to objects in the heap

Downcasts or specializing casts

- Most casts we have seen have been from a *derived* to a *base* object
  - These are called *upcasts* or *generalizing* casts
- In the TestA example, and the example on the right, we have a cast from a Base reference to a Derived reference
  - This is a specializing cast or down cast

```
class Test {
   public static void main(String args[]) {
      Derived d1 = new Derived( );

      Base b2 = d1;
      . . .
      ((Derived) b2).print2( );
   }
}
```

Unlike upcasts or generalizing casts downcasts can lead to errors when what is being referred to by the Base type is not the type of the cast or something derived from that type.

Downcasts or specializing casts

- Most if not all casts we have seen have been from a *derived* to a *base* object
  - These are called *upcasts* or *generalizing* casts
- In the TestA example, and the example on the right, we have a cast from a Base reference to a Derived reference
  - This is a specializing cast or down cast

```
class Test {
   public static void main(String args[]) {
      Derived d1 = new Derived( );

      Base b2 = d1;
      . . .
      ((Derived) b2).print2( );
   }
}
```

- In the case above *b2* refers to a base object which has no print2( ) defined in its VFT, thus no print2 exists to be called.

Example of bad *implicit* down casting

```
public class Base {
   public Base( ) { }
   public void print( ) {System.out.println("Base");}

}

public class Derived extends Base {
   public Derived( ) { }
   public void print( ) {System.out.println("Derived");}

}

public class Main {
   public static void main(String[] args) {
      Base b = new Base( );
      Derived d = new Derived( );
      b = d; // OK, Derived ISA Base
      d = b; // ILLEGAL! Base ISA not a Derived
    }
}
```

- Even though the Java compiler, in this case, could know
  - The object referenced by **d** is a Derived object
  - The **d** reference can legally point to a Derived object
- This is still illegal because for assignment **i = r**, it must be true that **r ISA i. This is a Java rule that you must follow**

Assume previous implicit down cast were allowed

What should happen here?

```
public class Base {
   public Base( ) { }
   public void print( ) {System.out.println("Base");}
   public int zero( ) {return 0;}

}

public class Derived extends Base {
   public Derived( ) { }
   public void print( ) {System.out.println("Derived");}

}

public class Main {
   public static void main(String[] args) {
      Base b = new Base( );
      Derived d = new Derived( );
      b = d; // Derived ISA Base
      d = b; // ILLEGAL! Base ISA not a Base
    }
}
```

- A smart compiler would figure out that at the red statement **b** references a Derived object and program would be legal.
- A dumb compiler would not know what **b** pointed to in the red statement and program would be illegal.
- Legality of the program would depend on the compiler.
- Kills portability and generally a bad thing to do.

Assume previous cast were allowed
What should happen here?

```
public class Base {
   public Base( ) { }
   public void print( ) {System.out.println("Base");}
}

public class Derived extends Base {
   public Derived( ) { }
   public void print( ) {System.out.println("Derived");}

}

public class Main {
   public static void main(String[] args) {
      Base b = new Base( );
      Derived d = new Derived( );
      b = d; // Derived ISA Base
      d = (Derived) b; // possible runtime test
    }
}
```

- This is legal but may require a runtime test.
- A smart compiler would figure out that at the red statement **b** references a Derived object and not do a runtime test
- A dumb compiler would not know what **b** pointed to in the red statement and do a runtime test.
- The cast indicates the programmer might have a clue and thus Java does a runtime test, if necessary, to ensure legality of the down cast.

Assume previous cast were allowed

What should happen here?

```
public class Base {
   public Base( ) { }
   public void print( ) {System.out.println("Base");}
}

public class Derived extends Base {
   public Derived( ) { }
   public void print( ) {System.out.println("Derived");}

}

public class Main {
   public static void main(String[] args) {
      Base b = new Base( );
      Derived d = new Derived( );
     if (foo( ) == 0) b = d;
      d = (Derived) b; // possible runtime test
    }
}
```

- This may or may not be legal, depending on the result of the *if* statement
- Doing a runtime test, as before, makes it all work because an error will be called if it is illegal and the program will run if it is legal.
- Unless you *know*, as a programmer, the downcast is legal, you should not do this
  - It is a rich source of errors that will only be caught at runtime
  - Embarrassing when it brings down Amazon or during a demo.

How to execute and run a Java program from a terminal window

```
smidkiffs-MacBook-Air:L1PolyOverride smidkiff$ ls
Pentagon.java Square.java TestAlt.java  spoor
Poly.java Test.java Triangle.java
smidkiffs-MacBook-Air:L1PolyOverride smidkiff$ javac Test.java
smidkiffs-MacBook-Air:L1PolyOverride smidkiff$ ls
Pentagon.class Poly.class  Square.class Test.class  TestAlt.java spoor
Pentagon.java Poly.java Square.java Test.java Triangle.java
smidkiffs-MacBook-Air:L1PolyOverride smidkiff$ java Test

output from the run

smidkiffs-MacBook-Air:L1PolyOverride smidkiff$
```

How not to compile a Java program from a terminal window

```
smidkiffs-MacBook-Air:L1PolyOverride smidkiff$ java Test.java
Exception in thread "main" java.lang.NoClassDefFoundError: Test/java
Caused by: java.lang.ClassNotFoundException: Test.java
  at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
  at java.security.AccessController.doPrivileged(Native Method)
  at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
  at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
  at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
  at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
smidkiffs-MacBook-Air:L1PolyOverride smidkiff$
```

The end