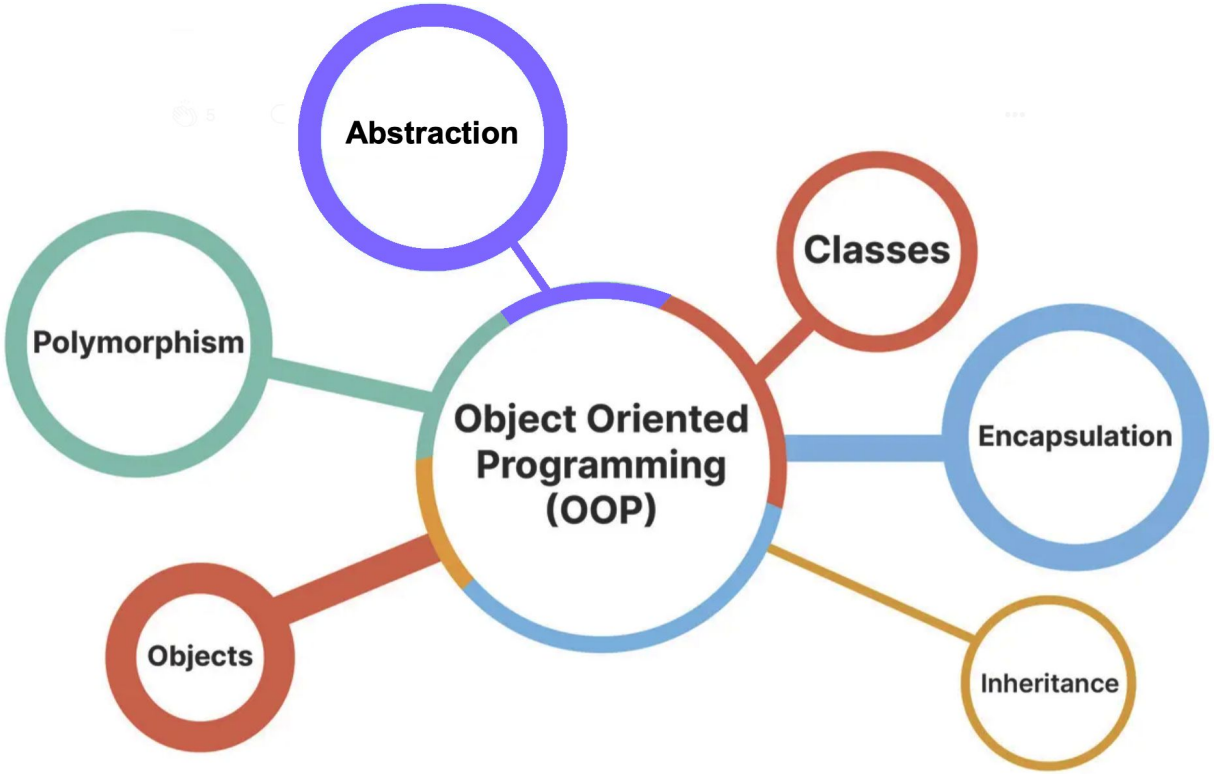
COSC 121: Computer Programming II

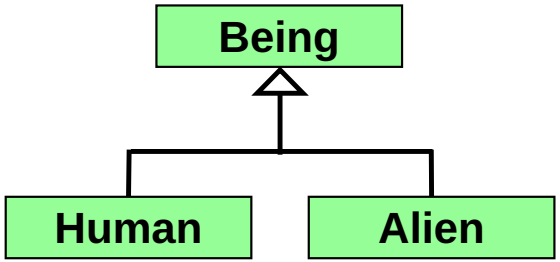


Today's Key Concepts

- Polymorphism is a new OOP technique for reference different object types that are related via inheritance or interface
- Implementation techniques:
  - Dynamic binding (as opposed to static binding)
  - The Three Rules
- Relationship to overriding
- Relationship to generic programming
- Using the instanceof operator
- Object casting (just like type casting)

Quick Review

```
public class Being {
    public void info() { System.out.println( "Being" ); }
}
public class Alien extends Being {
    public void info() { System.out.println( "Alien" ); }
}
public class Human extends Being {
    public void info() { System.out.println( "Human" ); }
}
```

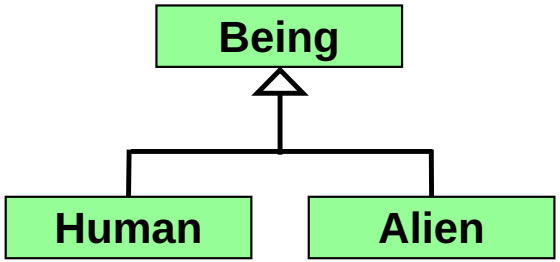


Output?

```
public class TestBeings {
    public static void main( String[] args )
    {
        Being b = new Being();
        Human h = new Human();
        Alien a = new Alien();
        b.info();
        b = h;
        b.info();
        b = a;
        b.info();
    }
}
```

Quick Review

```
public class Being {
    public void info() { System.out.println( "Being" ); }
}
public class Alien extends Being {
    public void info() { System.out.println( "Alien" ); }
}
public class Human extends Being {
    public void info() { System.out.println( "Human" ); }
}
```




```
public class TestBeings {
    public static void main( String[] args )
    {
        Being b = new Being();
        Human h = new Human();
        Alien a = new Alien();
        b.info();
        b = h;
        b.info();
        b = a;
        b.info();
    }
}
```

Output:
Being
Human
Alien

Dynamic versus Static Binding

- Recall dynamic binding (**late** binding):
  - Happens at run time
  - Program execution determines the correct method at runtime based on the actual object's class

Dynamic versus Static Binding

- Recall dynamic binding (**late** binding):
  - Happens at run time
  - Program execution determines the correct method at runtime based on the actual object's class
- Static binding (**early** binding):
  - Happens at compile time
  - Compiler determines the exact method to call using class/reference type
  - Does "method matching"

Dynamic versus Static Binding

- Recall dynamic binding (**late** binding):
  - Happens at run time
  - Program execution determines the correct method at runtime based on the actual object's class
- Static binding (**early** binding):
  - Happens at compile time
  - Compiler determines the exact method to call using class/reference type
  - Does "method matching"
- Java uses both:
  - Use static binding when compiler knows exactly which method to call at compile time
  - Use dynamic binding when compiler isn't sure which method to call (becomes JVM's job, enabling polymorphism)

When Have You Seen Static Binding?

When Have You Seen Static Binding?

- Overloading example:

```
public class Calculator {
    public int add( int a, int b ) {
        return a + b;
    }
    public int add( int a, int b, int c ) {
        return a + b + c;
    }
    public double add( double a, double b ) {
        return a + b;
    }
}
```

In main(), calling
calc.add(10, 20) vs.
calc.add(10, 20, 30) vs.
calc.add(3.4, 2.1)

- compiler knows which add() should be called in each case

Where Does Java Use Dynamic Binding?

- Polymorphism example:

```
public class Calculator {
    public int add( int a, int b ) {
        return a + b;
    }
}
public class TrickCalculator extends Calculator{
    public int add( int a, int b ) {
        return a - b;
    }
}
```

In main(), calling
calc.add(10, 20)

- confuses the compiler – what is the type of calc when method is called?

Where Does Java Use Dynamic Binding?

- Polymorphism example:

```
public class Calculator {
    public int add( int a, int b ) {
        return a + b;
    }
}
public class TrickCalculator extends Calculator{
    public int add( int a, int b ) {
        return a - b;
    }
}
```

In main(), calling
calc.add(10, 20)

- confuses the compiler – what is the type of calc when method is called?

```
Calculator calc = new Calculator();
System.out.println( calc.add( 10, 20 ) );
calc = new TrickCalculator();
System.out.println( calc.add( 10, 20 ) );
```

What About This?

- Consider this example:

```
public class Calculator {
    public int add( int a, int b ) {
        return a + b;
    }
    public double add( int a, int b ) {
        return a + b + 0.0;
    }
}
```

Same signature but different return types?

What About This?

- Consider this example:

```
public class Calculator {
    public int add( int a, int b ) {
        return a + b;
    }
    public double add( int a, int b ) {
        return a + b + 0.0;
    }
}
```

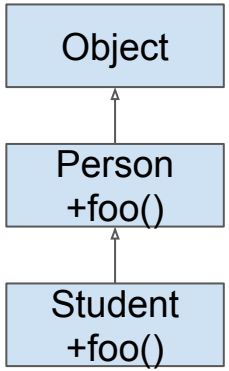
Same signature but different return types?

- Compiler error because Java requires method signatures to be unique, regardless of the return type

Longer Example Using Both Static and Dynamic Binding

```
public class Binding {
  public static void main( String[] args )
  {
      Person h = new Student();
      print( h );
  }
  static void print( Object x )
  {
      System.out.print("1: object " );
  }
  static void print( Person x )
  {
      System.out.print( "2: " );
      x.foo();
  }
  static void print( Student x )
  {
      System.out.print( "3: " );
      x.foo();
  }
}
```

```
class Person {
    void foo()
    {
        System.out.println( "person" );
    }
}
class Student extends Person {
    void foo()
    {
        System.out.println( "student" );
    }
}
```

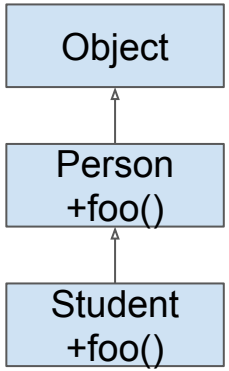


Output?

Longer Example Using Both Static and Dynamic Binding

```
public class Binding {
  public static void main( String[] args )
  {
      Person h = new Student();
      print( h );
  }
  static void print( Object x )
  {
      System.out.print("1: object " );
  }
  static void print( Person x )
  {
      System.out.print( "2: " );
      x.foo();
  }
  static void print( Student x )
  {
      System.out.print( "3: " );
      x.foo();
  }
}
```

```
class Person {
    void foo()
    {
        System.out.println( "person" );
    }
}
class Student extends Person {
    void foo()
    {
        System.out.println( "student" );
    }
}
```



Output?

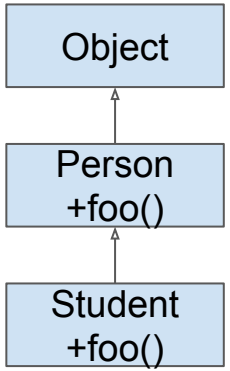
Trace:

- create h (a Student object)
- which print() is applicable to Person?

Longer Example Using Both Static and Dynamic Binding

```
public class Binding {
  public static void main( String[] args )
  {
      Person h = new Student();
      print( h );
  }
  static void print( Object x )
  {
      System.out.print("1: object " );
  }
  static void print( Person x )
  {
      System.out.print( "2: " );
      x.foo();
  }
  static void print( Student x )
  {
      System.out.print( "3: " );
      x.foo();
  }
}
```

```
class Person {
    void foo()
    {
        System.out.println( "person" );
    }
}
class Student extends Person {
    void foo()
    {
        System.out.println( "student" );
    }
}
```



Output: 2:

Trace:

- create h (a Student object)
- Java chooses most specific one allowed for the declared type: print(Person)

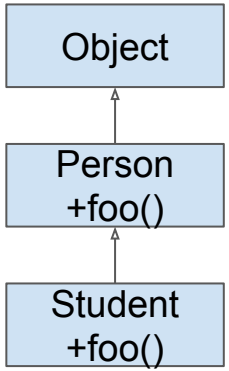
  end of static binding

  Java "binds" the call to static void print(Person x)

Longer Example Using Both Static and Dynamic Binding

```
public class Binding {
  public static void main( String[] args )
  {
      Person h = new Student();
      print( h );
  }
  static void print( Object x )
  {
      System.out.print("1: object " );
  }
  static void print( Person x )
  {
      System.out.print( "2: " );
      x.foo();
  }
  static void print( Student x )
  {
      System.out.print( "3: " );
      x.foo();
  }
}
```

```
class Person {
    void foo()
    {
        System.out.println( "person" );
    }
}
class Student extends Person {
    void foo()
    {
        System.out.println( "student" );
    }
}
```



Output: 2:

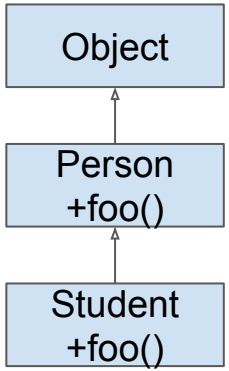
Trace:

- create h (a Student object)
- Java chooses most specific one allowed for the declared type: print(Person)
- which foo() are applicable to x ?

Longer Example Using Both Static and Dynamic Binding

```
public class Binding {
  public static void main( String[] args )
  {
      Person h = new Student();
      print( h );
  }
  static void print( Object x )
  {
      System.out.print("1: object " );
  }
  static void print( Person x )
  {
      System.out.print( "2: " );
      x.foo();
  }
  static void print( Student x )
  {
      System.out.print( "3: " );
      x.foo();
  }
}
```

```
class Person {
    void foo()
    {
        System.out.println( "person" );
    }
}
class Student extends Person {
    void foo()
    {
        System.out.println( "student" );
    }
}
```



Output: 2: student

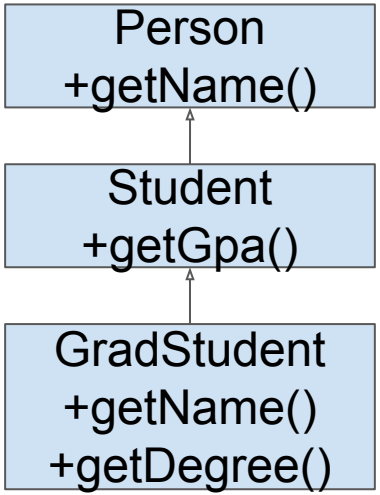
Trace:

- create h (a Student object)
- Java chooses most specific one: print(Person)
- x points to h, which is a Student object
- Student overrides foo()
- calls Student's implementation of foo()

Quick Review

- Is the following valid or invalid?

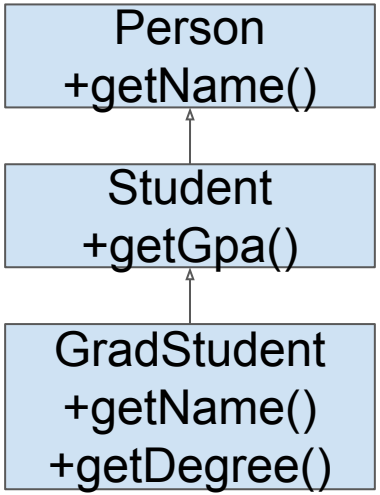
```
Person p = new GradStudent();
p.getName();
p.getDegree();
p.getGpa()
```



Quick Review

- Is the following valid or invalid?

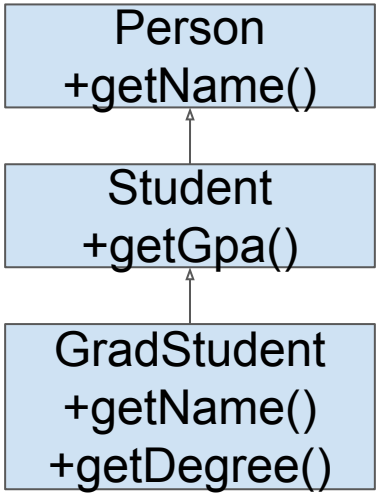
```
Person p = new GradStudent();        // valid
p.getName();
p.getDegree();
p.getGpa()
```



Quick Review

- Is the following valid or invalid?

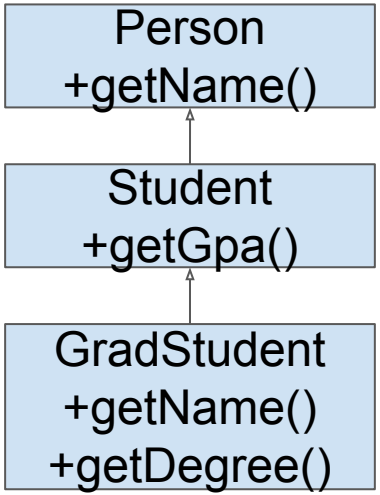
```
Person p = new GradStudent();        // valid
p.getName();                         // valid
p.getDegree();
p.getGpa()
```



Quick Review

- Is the following valid or invalid?

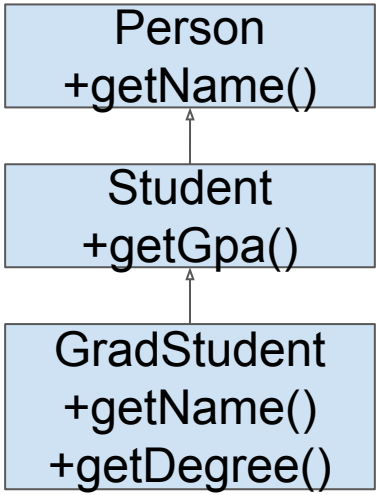
```
Person p = new GradStudent();      // valid
p.getName();                       // valid
p.getDegree();                     // invalid, R2
p.getGpa()
```



Quick Review

- Is the following valid or invalid?

```
Person p = new GradStudent();      // valid
p.getName();                       // valid
p.getDegree();                     // invalid, R2
p.getGpa();                        // invalid, R2
```

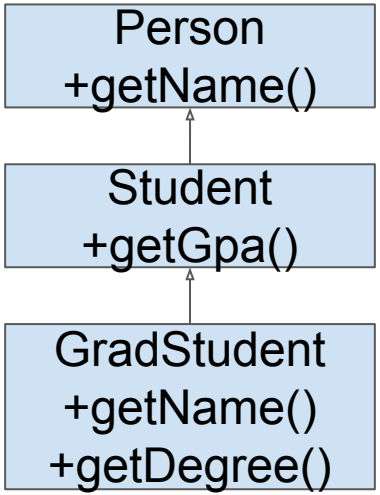


Declaring the most general type is not always the best approach!

Quick Review

- Is the following valid or invalid?

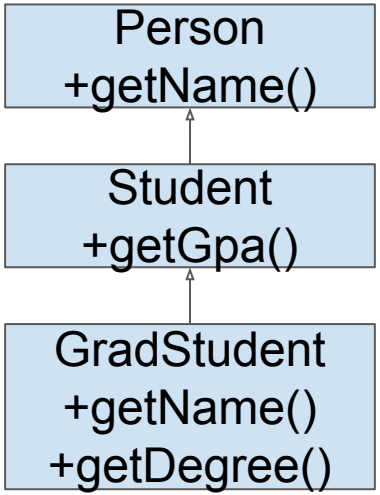
```
Student s = new GradStudent();
s.getName();
s.getDegree();
s.getGpa();
```



Quick Review

- Is the following valid or invalid?

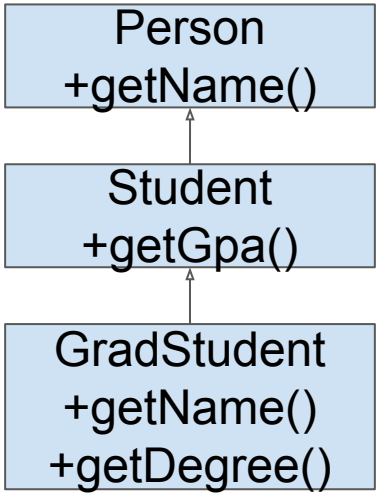
```
Student s = new GradStudent();        // valid
s.getName();
s.getDegree();
s.getGpa();
```



Quick Review

- Is the following valid or invalid?

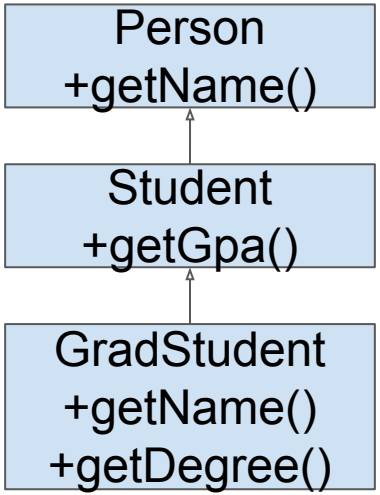
```
Student s = new GradStudent();      // valid
s.getName();                        // valid
s.getDegree();
s.getGpa();
```



Quick Review

- Is the following valid or invalid?

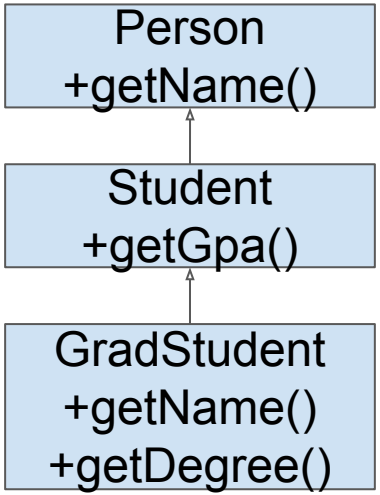
```
Student s = new GradStudent();      // valid
s.getName();                        // valid
s.getDegree();                      // invalid, R2
s.getGpa();
```



Quick Review

- Is the following valid or invalid?

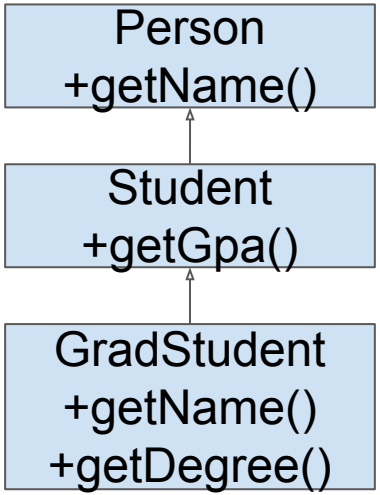
```
Student s = new GradStudent();      // valid
s.getName();                        // valid
s.getDegree();                      // invalid, R2
s.getGpa();                         // valid
```



Quick Review

- Is the following valid or invalid?

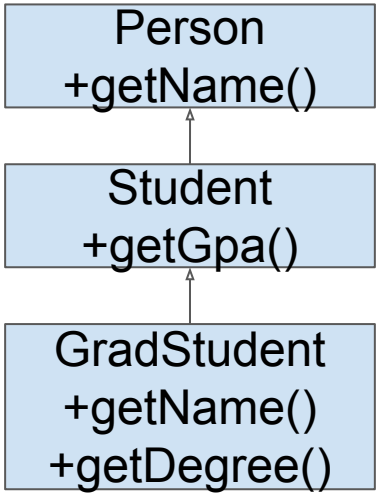
```
Student s = new Person();
s.getName();
s.getDegree();
s.getGpa();
```



Quick Review

- Is the following valid or invalid?

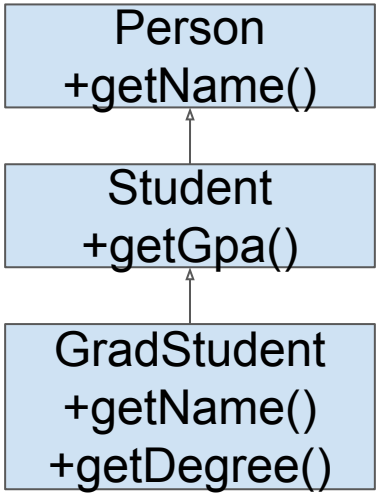
```
Student s = new Person();          // invalid, R1
s.getName();
s.getDegree();
s.getGpa();
```



Quick Review

- Is the following valid or invalid?

```
Student s = new Person();          // invalid, R1
s.getName();                       // invalid
s.getDegree();                     // invalid
s.getGpa();                        // invalid
```



iClicker Question

What is printed in the output?

A. Figure
   Figure
   Figure
B. Box
   Box
   Box
C. Figure
   Polygon
   Box
D. Error

```
public class Example {
  public static void main( String[] args )
  {
      Figure  f = new Figure();
      Polygon p = new Polygon();
      Box     b = new Box();
      f.display();
      f = p;
      f.display();
      f = b;
      f.display();
  }
}
```

```
class Figure {
    void display() { System.out.println( "Figure" ); }
}

class Polygon extends Figure {
    void display() { System.out.println( "Polygon" ); }
}

class Box extends Figure {
    void display() { System.out.println( "Box" ); }
}
```

iClicker Question

What is printed in the output?

A. Figure
   Figure
   Figure
B. Box
   Box
   Box
C. Figure
   Polygon
   Box
D. Error

```
public class Example {
  public static void main( String[] args )
  {
      Figure  f = new Figure();
      Polygon p = new Polygon();
      Box     b = new Box();
      f.display();
      f = p;
      f.display();
      f = b;
      f.display();
  }
}
```

```
class Figure {
    void display() { System.out.println( "Figure" ); }
}

class Polygon extends Figure {
    void display() { System.out.println( "Polygon" ); }
}

class Box {
    void display() { System.out.println( "Box" ); }
}
```

Determining the Object's Static Type

- Static type of an object is the type *known* at compile time

In general:

```
Type refVar = new ClassName();
```

static type of refVar

dynamic type

Determining the Object's Static Type

- Static type of an object is the type *known* at compile time
- Consider the following code:

```
Figure f1 = new Figure();
Figure f2 = new Polygon();
Figure f3 = new Box();
m( f1 );
m( f2 );
```

What is the type of the arguments passed into m()?

In general:

```
Type refVar = new ClassName();
```

static type of refVar

dynamic type

Determining the Object's Static Type

- Static type of an object is the type *known* at compile time
- Consider the following code:

```
Figure f1 = new Figure();
Figure f2 = new Polygon();
Figure f3 = new Box();
m( f1 );
m( f2 );                    // Figure, type of the reference variable
```

What is the type of the arguments passed into m()?

In general:
Type refVar = new ClassName();

static type of refVar

dynamic type

- What about this case?

```
m( new Box() );
```

Determining the Object's Static Type

- Static type of an object is the type *known* at compile time
- Consider the following code:

```
Figure f1 = new Figure();
Figure f2 = new Polygon();
Figure f3 = new Box();
m( f1 );
m( f2 );                    // Figure, type of the reference variable
```

What is the type of the arguments passed into m()?

In general:
Type refVar = new ClassName();

static type of refVar

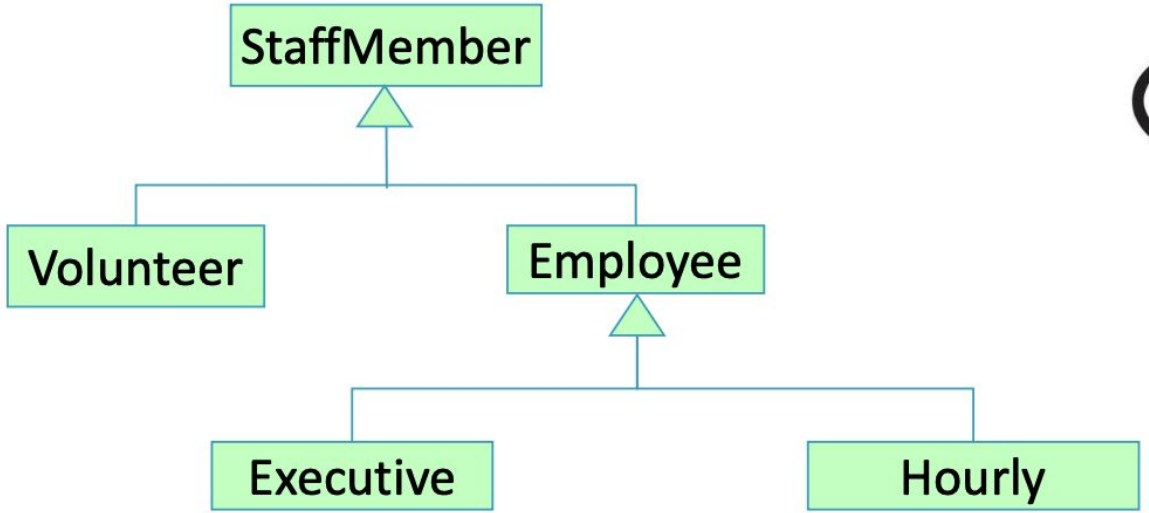
dynamic type

- What about this case?

```
m( new Box() );             // Box, due to using new
```

Programming Exercise (~5 min)

Create the classes represented in the following hierarchy. Create a test class that has an array of StaffMember objects that holds 2 volunteers, 1 executive member, and 2 hourly paid employees. Create a name attribute in each class and a corresponding accessor. Loop through the array and print everyone's names.



Type Checking

- Use the instanceof operator to check if an object is an instance of a class
- Suppose you have a reference variable but you are not sure the type of the object it points to
- Check if object x points to a specific class A:

```
if( x instanceof A )                    // part of the boolean expression
  System.out.println( "x is an instance of Class A" );
```

- Ex: check if x points to the Dog class:

```
if( x instanceof Dog )
  System.out.println( "x is a Dog" );
```

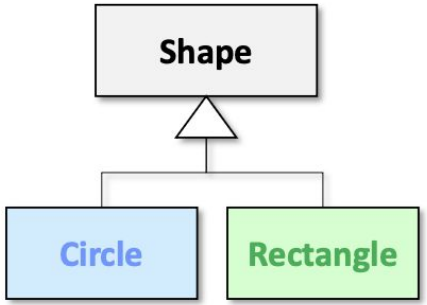
- Must have a target class in mind to check against

instanceof is also transitive

- An object is an instance of its own class and also of its ancestor classes
- Example:

  s instanceof Circle?
  c instanceof Circle?
  r instanceof Circle?
  s instanceof Shape?
  c instanceof Shape?
  r instanceof Shape?

```
Shape s = new Shape();
Shape c = new Circle();
Shape r = new Rectangle();
```



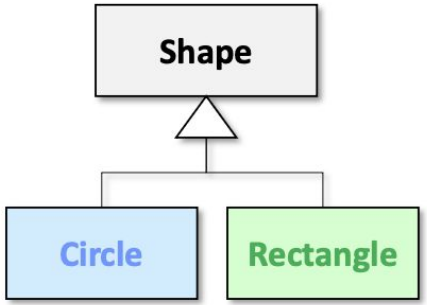
instanceof is also transitive

- An object is an instance of its own class and also of its ancestor classes
- Example:

  s instanceof Circle?
  c instanceof Circle?
  r instanceof Circle?
  s instanceof Shape?
  c instanceof Shape?
  r instanceof Shape?

❌

```
Shape s = new Shape();
Shape c = new Circle();
Shape r = new Rectangle();
```

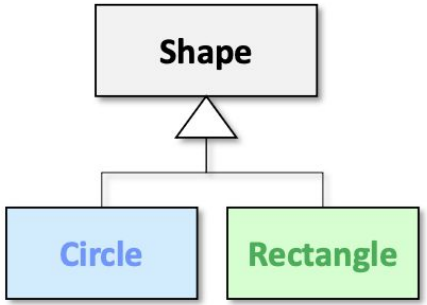


instanceof is also transitive

- An object is an instance of its own class and also of its ancestor classes
- Example:

s instanceof Circle? ❌
c instanceof Circle? ✔️
r instanceof Circle?
s instanceof Shape?
c instanceof Shape?
r instanceof Shape?

```
Shape s = new Shape();
Shape c = new Circle();
Shape r = new Rectangle();
```

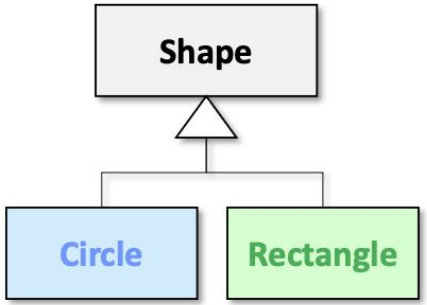


instanceof is also transitive

- An object is an instance of its own class and also of its ancestor classes
- Example:

s instanceof Circle? ❌
c instanceof Circle? ✔️
r instanceof Circle? ❌
s instanceof Shape?
c instanceof Shape?
r instanceof Shape?

```
Shape s = new Shape();
Shape c = new Circle();
Shape r = new Rectangle();
```

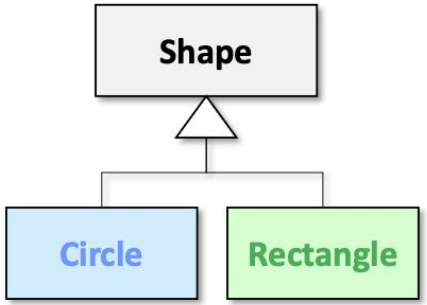


instanceof is also transitive

- An object is an instance of its own class and also of its ancestor classes
- Example:

s instanceof Circle? ❌
c instanceof Circle? ✔️
r instanceof Circle? ❌
s instanceof Shape? ✔️
c instanceof Shape?
r instanceof Shape?

```
Shape s = new Shape();
Shape c = new Circle();
Shape r = new Rectangle();
```

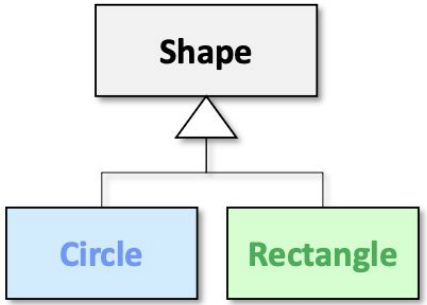


instanceof is also transitive

- An object is an instance of its own class and also of its ancestor classes
- Example:

s instanceof Circle? ❌
c instanceof Circle? ✔️
r instanceof Circle? ❌
s instanceof Shape? ✔️
c instanceof Shape? ✔️
r instanceof Shape?

```
Shape s = new Shape();
Shape c = new Circle();
Shape r = new Rectangle();
```

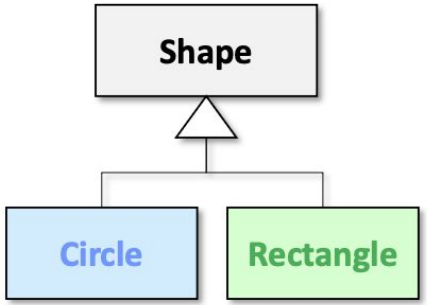


instanceof is also transitive

- An object is an instance of its own class and also of its ancestor classes
- Example:

s instanceof Circle? ❌
c instanceof Circle? ✔️
r instanceof Circle? ❌
s instanceof Shape? ✔️
c instanceof Shape? ✔️
r instanceof Shape? ✔️

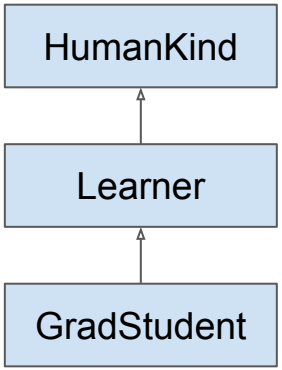
```
Shape s = new Shape();
Shape c = new Circle();
Shape r = new Rectangle();
```



iClicker Question

What is the output?

A. 1122
B. 1123
C. 2223
D. 2233
E. 3333



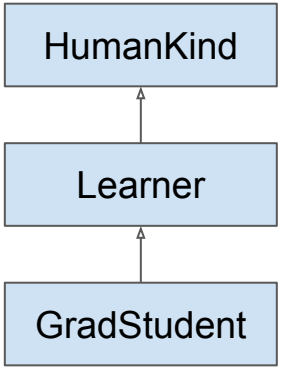

```
public class InstanceInheritance {
  public static void main( String[] args )
  {
    m( new GradStudent() );
    m( new Learner() );
    m( new HumanKind() );
    m( new Object() );
  }
  public static void m( Learner s ) { System.out.print(1); }
  public static void m( Object obj )
  {
    if( obj instanceof HumanKind )
      System.out.print(2);
    else
      System.out.print(3);
  }
}

class HumanKind{}
class Learner extends HumanKind{}
class GradStudent extends Learner{}
```

what are the types of these input arguments?

iClicker Question

What is the output?

A. 1122
B. 1123
C. 2223
D. 2233
E. 3333




```
public class InstanceInheritance {
  public static void main( String[] args )
  {
    Object o1 = new GradStudent();
    Object o2 = new Learner();
    Object o3 = new HumanKind();
    Object o4 = new Object();
    m( o1 );
    m( o2 );
    m( o3 );
    m( o4 );
  }
  public static void m( Learner s ) { System.out.print(1); }
  public static void m( Object obj )
  {
    if( obj instanceof HumanKind )
      System.out.print(2);
    else
      System.out.print(3);
  }
}

class HumanKind{}
class Learner extends HumanKind{}
class GradStudent extends Learner{}
```

what is the type of o1, …, o4 at compile time?

Recall Casting

- Type casting (widening) example:

```
int myInt = 9;
double myDouble = myInt;          // Implicit casting: int ⇒ double
```

- Example in the opposite direction (narrowing):

```
double myDouble = 9.78d;
int myInt = ( int )myDouble;      // Explicit casting: double ⇒ int
```

Recall Casting

- Type casting (widening) example:

```
int myInt = 9;
double myDouble = myInt;          // Implicit casting: int ⇒ double
```

- Example in the opposite direction (narrowing):

```
double myDouble = 9.78d;
int myInt = ( int )myDouble;      // Explicit casting: double ⇒ int
```

- Explicit casting is dangerous
  - It forces the compiler to override its normal type-checking rules
  - Possible problems: data loss, runtime errors, and undefined behavior
  - Mechanism enables programmer to assert "I know what I'm doing, just do it"

Object Casting

- Implicit object casting happens as part of Rule 1:

```
Object obj = new Student();
Student s = obj;                                    // error: Not every object is a student
```

Object Casting

- Implicit object casting happens as part of Rule 1:

```
Object obj = new Student();
Student s = obj;                    // error: Not every object is a student
```

- With explicit object casting:

```
Object obj = new Student();
Student s = ( Student )obj;         // do this only if you truly know obj
                                    // really is a Student object
```

Object Casting

- Implicit object casting happens as part of Rule 1:

```
Object obj = new Student();
Student s = obj;                    // error: Not every object is a student
```

- With explicit object casting:

```
Object obj = new Student();
Student s = ( Student )obj;         // do this only if you truly know obj
                                    // really is a Student object
```

- Results in runtime error with ClassCastException if superclass is not an instance of the subclass:

```
Object obj = new Object();
Student s = ( Student )obj;         // no compile/syntax error
```

Examples to Consider

```
class Video {}
class Reel extends Video {}

class TestVideo
{
    public static void main( String[] args )
    {
        Video v1 = new Video();
        Video v2 = new Reel();

        Reel r = v2;                    // does this work?
    }
}
```

Examples to Consider

```
class Video {}
class Reel extends Video {}

class TestVideo
{
    public static void main( String[] args )
    {
        Video v1 = new Video();
        Video v2 = new Reel();

        Reel r = v2;                    // does this work? ❌
    }                                   need explicit casting
}
```

Examples to Consider

```
class Video {}
class Reel extends Video {}

class TestVideo
{
    public static void main( String[] args )
    {
        Video v1 = new Video();
        Video v2 = new Reel();

        Reel r = ( Reel )v2;                    // does this work?
    }
}
```

Examples to Consider

```
class Video {}
class Reel extends Video {}

class TestVideo
{
    public static void main( String[] args )
    {
        Video v1 = new Video();
        Video v2 = new Reel();

        Reel r = ( Reel )v2;                    // does this work? ✔️
    }
}
```

Examples to Consider

```
class Video {}
class Reel extends Video {}

class TestVideo
{
    public static void main( String[] args )
    {
        Video v1 = new Video();
        Video v2 = new Reel();

        Reel r = ( Reel )v1;                    // does this work?
    }
}
```

Examples to Consider

```
class Video {}
class Reel extends Video {}

class TestVideo
{
    public static void main( String[] args )
    {
        Video v1 = new Video();
        Video v2 = new Reel();

        Reel r = ( Reel )v1;
    }
}
```

// does this work? ❌

compiles fine but runtime ClassCastException

Examples to Consider

```
class Video {}
class Reel extends Video {}

class TestVideo
{
    public static void main( String[] args )
    {
        Video v1 = new Video();
        Video v2 = new Reel();

        Reel r = ( Reel )( new Video() );          // does this work?
    }
}
```

Examples to Consider

```
class Video {}
class Reel extends Video {}

class TestVideo
{
    public static void main( String[] args )
    {
        Video v1 = new Video();
        Video v2 = new Reel();

        Reel r = ( Reel )( new Video() );          // does this work? ❌
    }                                              same
}
```

Examples to Consider

```
class Video {}
class Reel extends Video {}

class TestVideo
{
    public static void main( String[] args )
    {
        Video v1 = new Video();
        Video v2 = new Reel();

        Video v3 = v1;                    // does this work?
    }
}
```

Examples to Consider

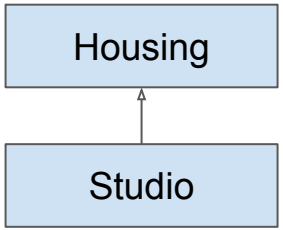
```
class Video {}
class Reel extends Video {}

class TestVideo
{
    public static void main( String[] args )
    {
        Video v1 = new Video();
        Video v2 = new Reel();

        Video v3 = v1;                    // does this work? ✔️
    }
}
```

iClicker Question

Which statement causes a compiler error if placed in the main() after the comment // … ?

A. Housing h = h1;
B. Studio s = h2;
C. Studio s = (Studio)h2;
D. Studio s = (Studio)h1;
E. Studio s =
(Studio)(new Housing());

```
public class CastingQuestion {
  public static void main( String[] args )
  {
      Housing h1  = new Housing();
      Housing h2  = new Studio();
      // ...
  }
}

class Housing {}
class Studio extends Housing {}
```

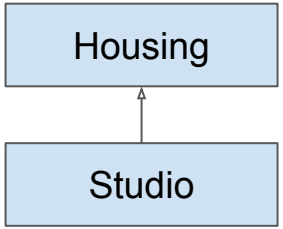


iClicker Question

Which statement causes a compiler error if placed in the main() after the comment // … ?

A. Housing h = h1;
B. Studio s = h2;
C. Studio s = (Studio)h2;
D. Studio s = (Studio)h1;
E. Studio s =
(Studio)(new Housing());

```
public class CastingQuestion {
  public static void main( String[] args )
  {
      Housing h1  = new Housing();
      Housing h2  = new Studio();
      // ...
  }
}

class Housing {}
class Studio extends Housing {}
```



D and E cause a ClassCastException error