Comp 401
Inheritance: Type Checking

Instructor: Prasun Dewan

Prerequisite

- Inheritance

Type-Checking Examples

```
StringHistory stringHistory = new AStringDatabase();
```

```
StringDatabase stringDatabase = new AStringHistory();
```

Type-Checking Examples

StringDatabase stringDatabase = **new** AStringHistory();

stringDatabase.clear()

StringHistory stringHistory = **new** AStringDatabase();

stringHistory.size()

stringHistory .clear()

((StringDatabase) stringHistory) .clear()

Type-Checking Examples

StringDatabase stringDatabase = **new** AStringHistory();

stringDatabase.clear()

StringHistory stringHistory = **new** AStringDatabase();

stringHistory.size()

stringHistory .clear()

((StringDatabase) stringHistory) .clear()

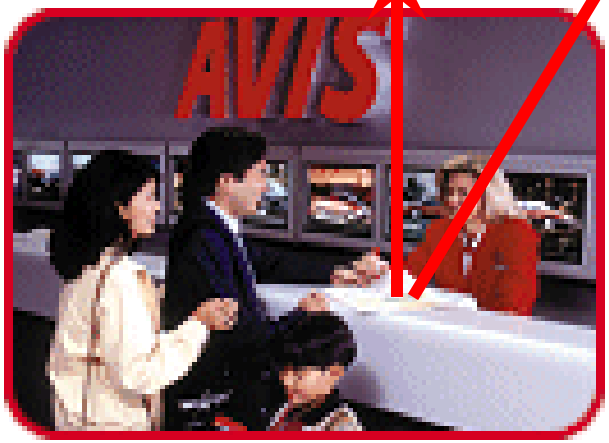
Car analogy?

Getting an Upgrade

RegularModel myCar = **new** ADeluxeModel ();

Regular Model Requested

Deluxe Model Assigned



myCar.steer();

myCar.
setCity(“Raleigh”);

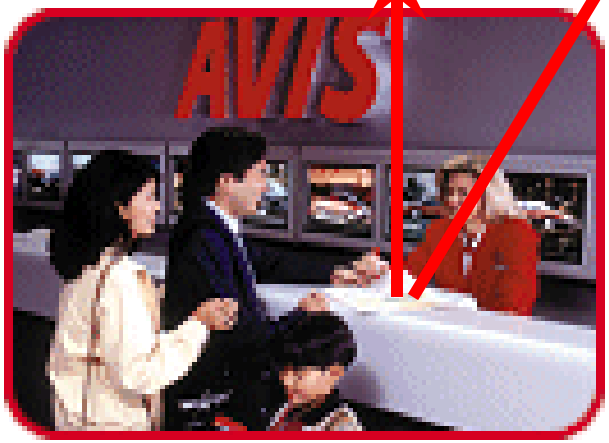
((DeluxeModel) myCar).
setCity(“Raleigh”);

Getting a Downgrade

DeluxeModel myCar = **new** ARegularModel ();

Deluxe Model Requested

Regular Model Assigned



myCar.steer();

myCar.
setCity("Raleigh"
);

Typing Arrays Elements

```
Object[] objects = { “Joe Doe”, new AStringDatabase(), new AStringHistory()};
```

```
String[] strings = {“Joe Doe”, new Object()};
```

Given an array of type T[], the type of each element of the array IS-A T.

The elements of an array can be of different types

Compile Time vs. Runtime Cast Errors: Casting newly constructed objects

```
StringHistory stringHistory = new AStringHistory();
….
StringDatabase database = (StringDatabase) stringHistory;
```

```
StringDatabase database = (StringDatabase) new AStringHistory();
```

Runtime, not compile time error

Cast done at runtime even for instantiated classes in Java 1.6

Compile Time vs. Runtime Cast Errors: Casting Classes

```
ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
ACartesianPoint cartesianPoint = (ACartesianPoint) bmiSpreadshee
```

bmiSpreadsheet can be assigned only an object of subtype of ABMISpreadsheet

No subtype of ABMISpreadsheet can be a subtype also of ACartesianPoint as Java has no multiple class inheritance

Can cast an object /variable of class C1 to Class C2 only if C1 is the same or super or subtype of C2

```
AStringHistory stringHistory = (AStringHistory) new AStringHistory();
AStringDatabase database = (AStringDatabase) stringHistory;
stringHistory = (AStringHistory) stringDatabase;
```

Compile Time vs. Runtime Cast Errors: Casting Classes (Review)

```
ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
ACartesianPoint cartesianPoint = (ACartesianPoint) bmiSpreadshee
```

bmiSpreadsheet can be assigned only an object of subtype of ABMISpreadsheet

No subtype of ABMISpreadsheet can be a subtype also of ACartesianPoint as Java has no multiple class inheritance

Can cast an object /variable of class C1 to Class C2 only if C1 is the same or super or subtype of C2

```
AStringHistory stringHistory = (AStringHistory) new AStringHistory();
AStringDatabase database = (AStringDatabase) stringHistory;
stringHistory = (AStringHistory) stringDatabase;
```

Compile Time vs. Runtime Cast Errors: Interfaces

```
ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
Point cartesianPoint = (Point) bmiSpreadsheet;
bmiSpreadsheet = (ABMISpreadsheet) cartesianPoint;
```

Some subtype of ABMISpreadsheet may implement Point

```
public class ABMISpreadsheetAndPoint
                extends ABMISpreadsheet
                implements Point {
…
}
```

```
ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheetAndPont();
Point cartesianPoint = (Point) bmiSpreadsheet;
bmiSpreadsheet = (ABMISpreadsheet) cartesianPoint;
```

Final Class

```
String string = "hello";
Point cartesianPoint = (Point) string;
```

Some subtype of String may implement Point?

```
public class AStringAndPoint
                extends String
                implements Point {
…
}
```

String is a final class and thus cannot be subtyped

```
public final class String {
…
}
```

Interface Casting Rules

Can cast an object /variable typed by a non final object type to any interface

Can cast an object /variable typed by an interface to any non final object type

Assignment Rules for Object Types

- If T1 IS-A T2, Expression of type T1 can be assigned to Variable of type T2
- Expression of type T1 can be assigned to Variable of type T2 with (legal) cast of (T1)

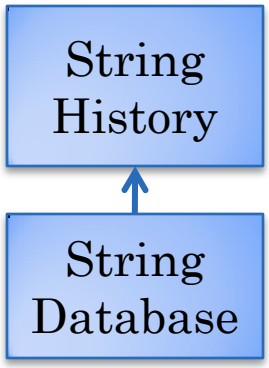
IS-A Rules Revisited

- Extends: T1 extends T2 => T1 IS-A T2
- Implements: T1 implements T2 => T1 IS-A T2
- Transitive:
  - T1 IS-A T2
  - T2 IS-A T3
  - => T1 IS-A T3
- Reflexive:
  - T1 == T2 => T1 IS-A T2

```
StringHistory stringHistory = new AStringDatabase();
Object o = stringHistory;
```

Special rule saying very type IS-A Object

Conceptual Problems in Java



println takes argument of type Object

! (Interface subtype of Class)

Interface subtype of Object

```
StringHistory stringHistory = new AStringHistory();
System.out.println(stringHistory);
```

Assigning an interface to an Object

Additional IS-A rule: T IS-A Object, for all T

Type-Checking Examples For Primitive Types

```
int i= 2.5;
```

```
double d = 2;
```

Assignment Rules for Primitive Types

- If T1 narrower than T2 (Set of instances of T1 $\subseteq$ Set of instances of T2)
- Expression of type T1 can be assigned to variable of type T2
- Expression of type T2 can be assigned to variable of type T1 with cast

Casting vs. Instance Of

((StringDatabase) stringHistory) .clear()

```
if (stringHistory instanceof StringDatabase) {
   ((StringDatabase) stringHistory) .clear();
} else {
   System.out.println("Got unlucky");
   System.exit(-1);
}
```

O **instanceof** T

Return true if Class of O IS-A T

If it is going to give up and terminate, then instanceof check duplicates the same check made by the cast to throw exception

Usually considered bad programming to decide alternatives based on instanceof.

If program has an alternative plan then use instanceof

Illustrating Instanceof

```
public void printPointOrBMIProperties(Object pointOrBMI) {
   if (pointOrBMI instanceof Point) {
     Point point = (Point) pointOrBMI;
     System.out.println(point.getX());
   } else if (pointOrBMI instanceof BMISpreadsheet) {
     BMISpreadsheet bmiSpreadsheet = (BMISpreadsheet) pointOrBMI;
     System.out.println(bmiSpreadsheet.getBMI());
   }
 }
```

Bad use of Instance Of

```
StringHistory stringHistory;
```

```
if (stringHistory instanceof StringDatabase) {
   sort((StringDatabase) stringHistory);
} else if (stringHistory instanceof StringSet) {
   sort((StringSet) stringHistory);
}
```

Usually considered bad programming to decide alternatives based on instanceof. Make the alternative actions implementations of the same method in the type of the variable used in instanceof

Declare sort() in type StringHistory

```
stringHistory.sort();
```

Do not have to write the messy if and change it when additional subtypes of the variable type are added

Good use of Instanceof

```
StringHistory stringHistory;
```

```
if (stringHistory instanceof StringDatabase) {
   display((StringDatabase) stringHistory);
} else if (stringHistory instanceof StringSet) {
   display((StringSet) stringHistory);
}
```

```
stringHistory.display()
```

- Display and data should be separate
- Sometimes separation of concerns requires instanceof
- Use instanceof for testing the class of tokens
- Scanning and parsing are separate

ObjectEditor and Inheritance

```
@StructurePattern(StructurePatternNames.POINT_PATTERN)
public interface Point {…}
```

```
@StructurePattern(StructurePatternNames.OVAL_PATTERN)
public class ACartesianPoint implements Point {…}
```

```
@StructurePattern(StructurePatternNames.RECTANGLE_PATTERN)
public class ABoundedPoint extends ACartesianPoint
implements Point {…}
```

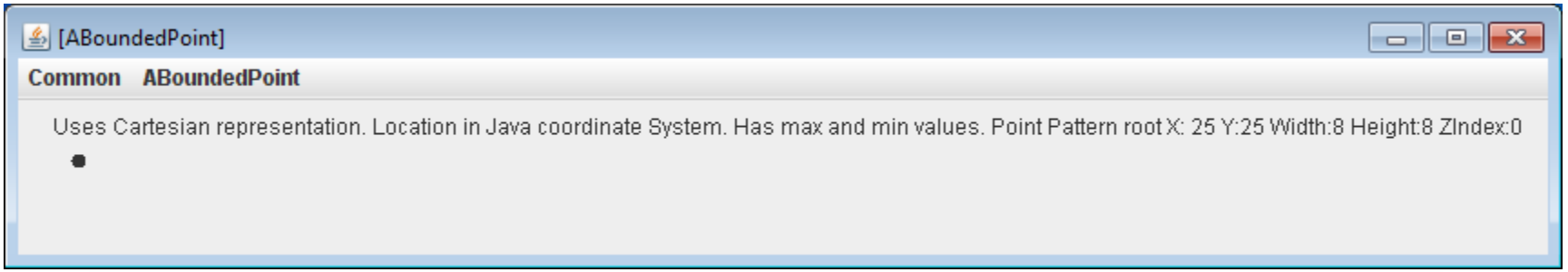
Class of the object gets precedence over its super types

Multiple Inheritance

```
@Explanation("Location in Java coordinate System.")
public interface Point {…}
```

```
@Explanation("Uses Cartesian representation.")
public class ACartesianPoint implements Point {…}
```

```
@Explanation("Has max and min values.")
public class ABoundedPoint extends ACartesianPoint
implements Point {…}
```



Sometimes inherited attributes can be combined rather than overridden.

Calling super() essentially achieves that and some languages automatically called overridden methods in addition to constructors

Extra Slides

Database

```
AStringDatabase [Java Application] C:\Program F
James Dean
Joe Doe
Jane Smith
p
********************
James Dean
Joe Doe
Jane Smith
********************
m Joe Doe
true
m Jane Doe
false
d Joe Doe
p
********************
James Dean
Jane Smith
********************
c
p
********************
********************
```

How does program distinguish between different input characters?

ObjectEditor and IS-A

```
public class ABoundedObject extends ACartesianPoint
                            implements Point {
  Point upperLeftCorner, lowerRightCorner;
  public ABoundedPoint(int initX, int initY,
          Point initUpperLeftCorner, Point
initLowerRightCorner) {
    super(initX, initY);
    x = Math.min(x, upperLeftCorner.getX());
    x = Math.max(x, lowerRightCorner.getX());
    y = Math.min(y, upperLeftCorner.getY());
    y = Math.max(y, lowerRightCorner.getY());
  }
}
```

Class does not have term Point

Supertypes ACartesianPoint and Point do

Given class C, OE looks at names of all T, C IS-A T

Nested IFs

```
public static void main(String args[]) {
        StringDatabase names = new AStringDatabase();
        while (true) {
            String input = System.console().readLine();
            if (!(input.length() == 0))
                if (input.charAt(0) == 'q')
                    break;
                else if (input.charAt(0) == 'p')
                     print(names);
                else if (input.charAt(0) == 'd')
                   names.deleteElement(input.substring(2, input.length()));
                else if (input.charAt(0) == 'm')
                   System.out.println(names.member(
                      input.substring(2, input.length())));
                else if (input.charAt(0) == 'c')
                   names.clear();
                else
                   names.addElement(input);
        }
}
```

Switch

```
public static void main(String args[]) {
        StringDatabase names = new AStringDatabase();
        while (true) {
            String input = System.console().readLine();
            if (!(input.length() == 0))
                if (input.charAt(0) == 'q')
                        break;
                 else switch (input.charAt(0)) {
                        case 'p':
                            print(names);
                            break;
                        case 'd':
                            names.deleteElement(input.substring(2, input.length()));
                            break;
                        case 'm':
                            System.out.println(names.member(input.substring(2, input.length())));
                            break;
                        case 'c':
                            names.clear();
                            break;
                        default:
                            names.addElement(input);
        }
}
```

Breaks out of the loop

Breaks out of the switch

Switch expression

Switch arm

Switch case (value of switch expression)

Multi-case Arms

```
public static void main(String args[]) {
    StringDatabase names = new AStringDatabase();
    while (true) {
        String input = System.console().readLine();
        if (!(input.length() == 0))
            if (input.charAt(0) == 'q')
                    break;
            else switch (input.charAt(0)) {
                    case 'p', 'P':
                        print(names);
                        break;
                    case 'd', 'D':
                        names.deleteElement(input.substring(2, input.length()));
                        break;
                    case 'm', 'M':
                        System.out.println(names.member(input.substring(2, input.length())));
                        break;
                    case 'c', 'C':
                        names.clear();
                        break;
                    default:
                        names.addElement(input);
        }
}
```

Omitting Break

```
public static void main(String args[]) {
        StringDatabase names = new AStringDatabase();
        while (true) {
            String input = System.console().readLine();
            if (!(input.length() == 0))
                if (input.charAt(0) == 'q')
                        break;
                 else switch (input.charAt(0)) {
                      case 'p', 'P':
                            print(names);
                      case 'd', 'D':
                            names.deleteElement(input.substring(2, input.length()));
                      case 'm', 'M':
                            System.out.println(names.member(input.substring(2, input.length())));
                     case 'c', 'C':
                            names.clear();
                     default:
                            names.addElement(input);
            }
}
```

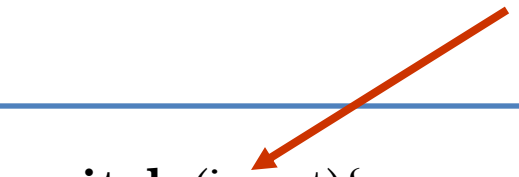
Code falls through to the next case arm without break

Switch expression evaluated once.

More efficient than nested ifs.

Illegal Switch

Type of switch expression must be ordinal type



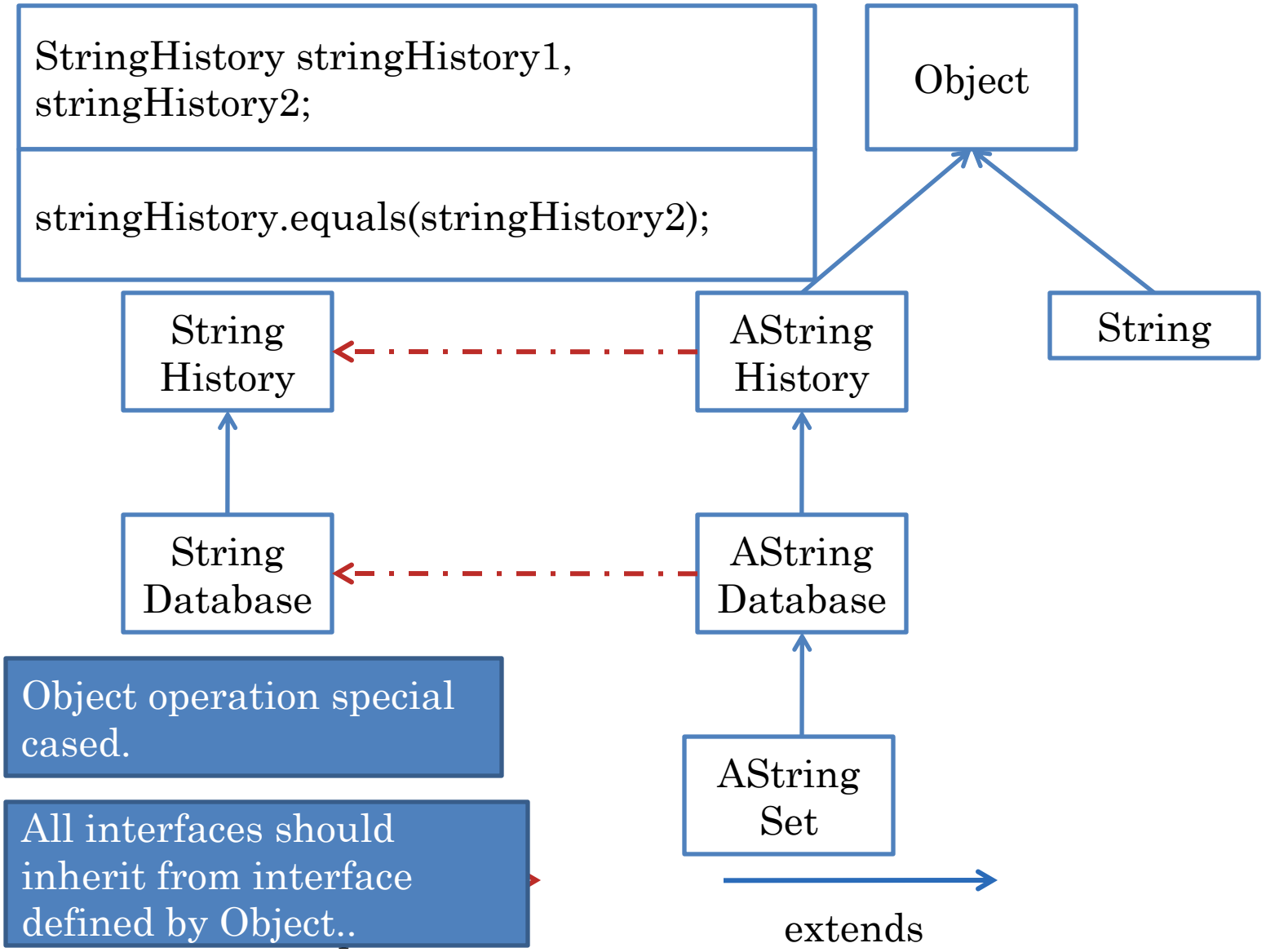
```
switch (input){
        case "print":
                print(names);
         case "clear":
                names.clear();
        default:
                names.addElement(input);
}
```

Ordinal Type

- Values of type are ordered.
- Each value has a unique successor and predecessor

char String int double

Object Operations on Variables Typed by Interfaces

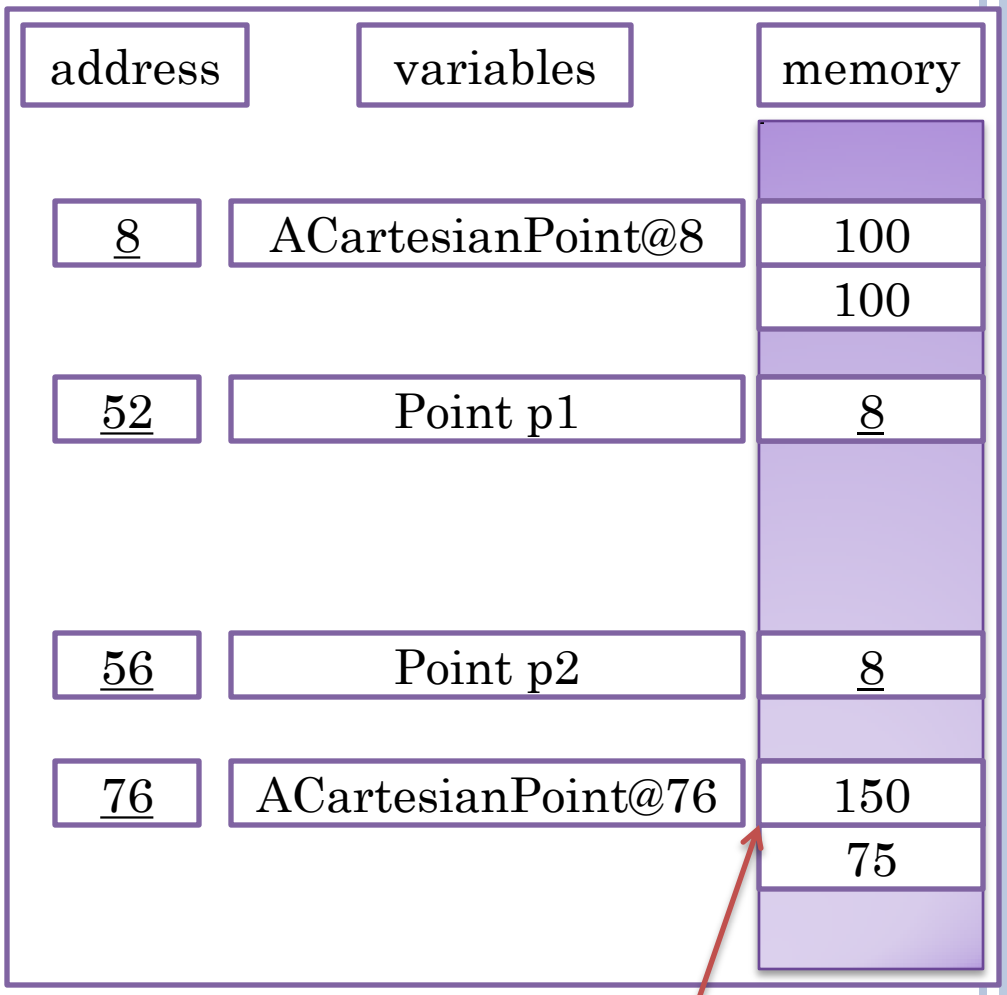


Garbage Collection

```
Point p1  = new
ACartesianPoint(100,100);
```

```
Point p2  = new
ACartesianPoint(150,75);
```

```
p2  = p1 ;
```



ObjectEditor Conventions And Inheritance

- Conventions for how methods are named for stateful, geometric, collection objects
- Makes it easier to find opportunities for inheritance
- However, not all methods defined by you are constrained by ObjectEditor conventions

Compile Time Errors

StringDatabase database= (StringDatabase) **new** AStringHistory();

String[] strings= (String) "Joe Doe";

```
Point point = (Point) (new AStringHistory());
```

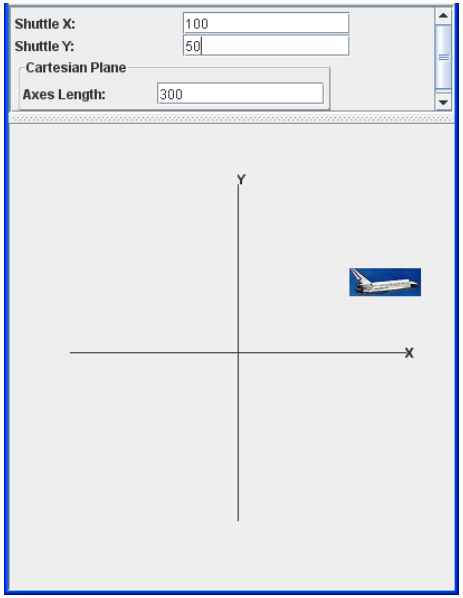
```
point = (Point) "hello";
```

Cast done at runtime even for instantiated classes in Java 1.6

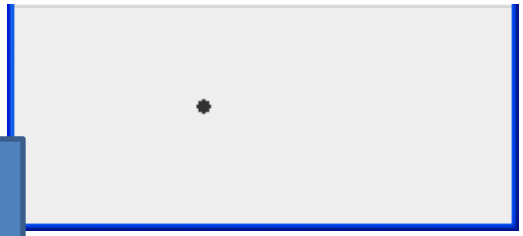
Can cast an object /variable of class C to type T only if C is the same or super or subtype of T

ObjectEditor Graphics Rules

```
public interface ShuttleLocation {
  public FancyCartesianPlane
        getCartesianPlane();
  public ImageLabel getShuttleLabel();
  public int  getShuttleX();
  public void setShuttleX(int newVal);
  public int  getShuttleY();
  public void setShuttleY(int newVal);
}
```




```
public interface NotAPoint {
  public FancyCartesianPlane
        getCartesianPlane();
  public ImageLabel getShuttleLabel();
  public int  getX();
  public void setX(int r
  public int  getY();
  public void setY(int r
  public Point getLocati
}
```



Type interpreted as a Point as its name contains "Point" and has X and Y Properties

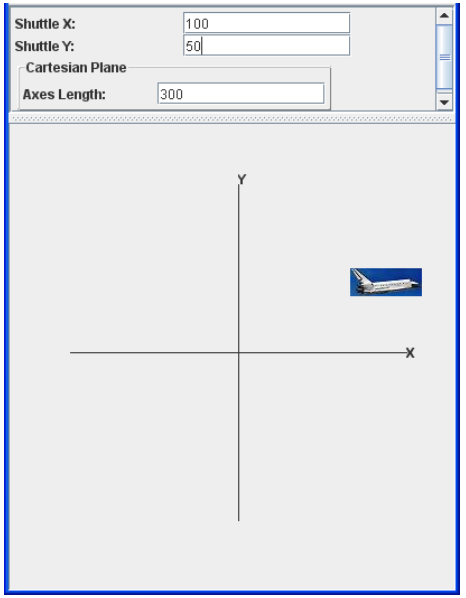
ObjectEditor Point Rules

- An object is recognized as a point representation if:
  - Its interface or class has the string "Point" in its name or has a Point annotation
  - It has (read-only) int properties, X and Y, representing Cartesian window coordinates
  - Can have additional properties

```
@StructurePattern(StructurePatternNames.POINT_PATTERN)
public interface Point {
        public int getX();
        public int getY();
        public double getAngle();
        public double getRadius();
}
```

IS_ATOMIC_SHAPE CLASS ANNOTATION

```
import util.annotations.IsAtomicShape;
@IsAtomicShape(false)
//same as AShuttleLocation except interface name
public class AShuttleLocationImplementingABadlyNamedInterface
                              implements NotAPoint {
```



IsAtomicShape(false) before class name says do not interpret the class as an atomic shape (it can be a composition)

Calling Constructor in Superclass

```
public class ABoundedPoint extends ACartesianPoint
                           implements Point {
  Point upperLeftCorner, lowerRightCorner;
  public ABoundedPoint(int initX, int initY,
          Point initUpperLef                    rner) {
    super(initX, initY);
    x = Math.max(x, upperLef
    x = Math.min(x, lowerRig
    y = Math.max(y, upperLef
    y = Math.min(y, lowerRig
  }
}
```

Slide to be deleted

Variables also extended unlike in collection example

Superclass constructor

Compile Time vs. Runtime Cast Errors: Casting Classes

```
ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
Point cartesianPoint = (Point) bmiSpreadsheet;
bmiSpreadsheet = (ABMISpreadsheet) cartesianPoint;
```

Some subtype of ABMISpreadsheet may implement Point

Casting Class to Class

```
ACartesianPoint cartesiaPoint = (ACartesianPoint) new ABMISpreadsheet();
```

Cast done at runtime even for instantiated classes in Java 1.6

Can cast an object /variable of class C to type T only if C is the same or super or subtype of T

Compile Time Errors

```
StringDatabase database= (StringDatabase) new AStringHistory();
```

```
String[] strings= (String)  “Joe Doe”;
```

```
Point point = (Point) (new AStringHistory());
```

```
point = (Point) "hello";
```

Cast done at runtime even for instantiated classes in Java 1.6

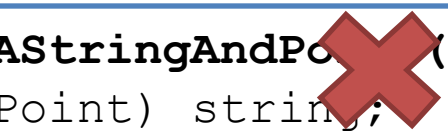
Can cast an object /variable of class C to type T only if C is the same or super or subtype of T

Casting with Interface for Instances of Final and Non-Final Class

```
public class AStringHistoryAndPoint
             extends AStringHistory implements Point { … }
```

```
StringHistory stringHistory = new AStringHistoryAndPoint();
Point point = (Point) stringHistory;
```

```
String s = new AStringAndPo          ();
Point point = (Point) strin          ;
```



```
public class AS          gAndPoint
             ext         String implements Point { … }
```

```
public final class String {…}
```