Comp 401
Inheritance: IS-A

Instructor: Prasun Dewan

Prerequisite

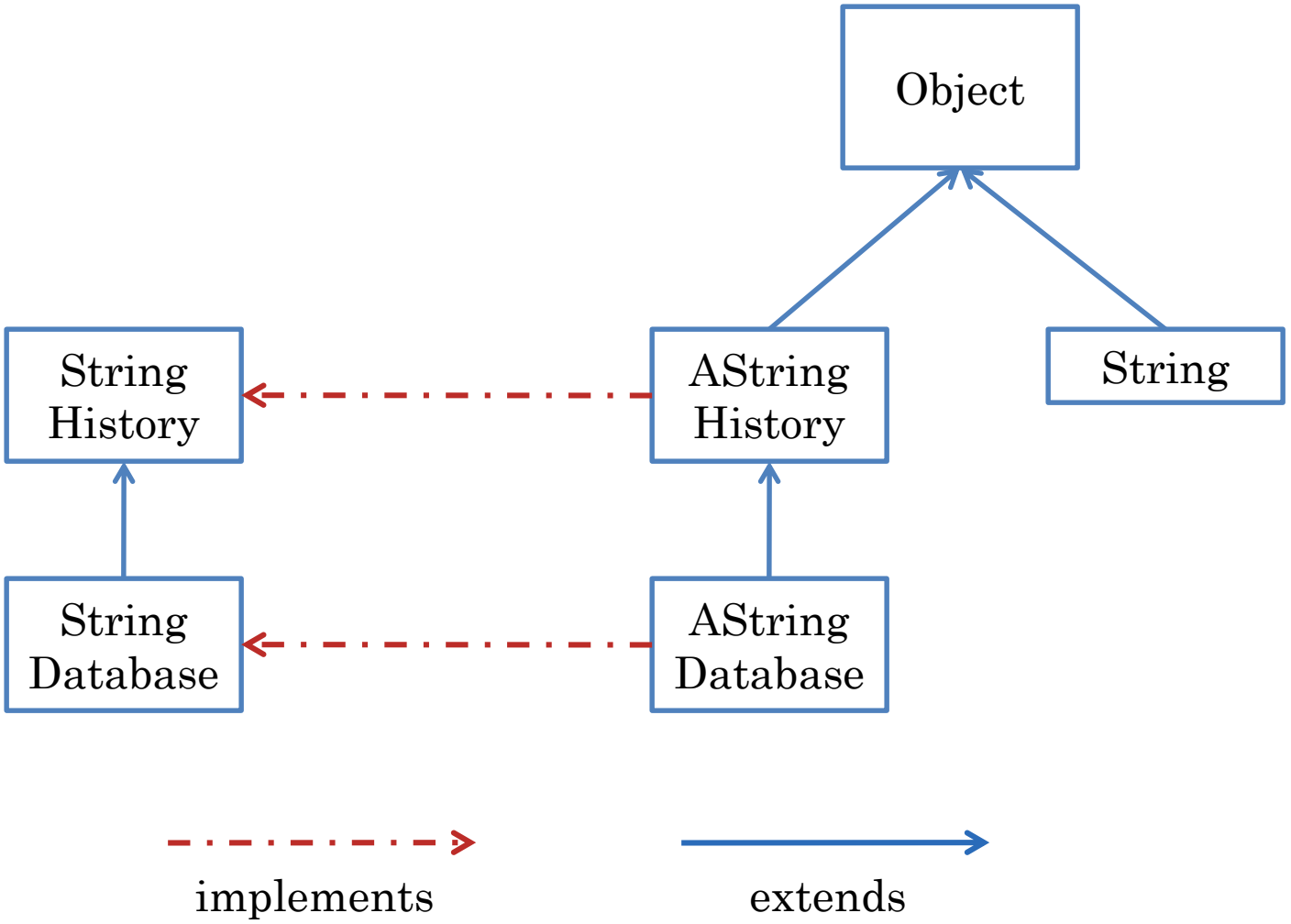
- Interfaces
- Inheritance and Arrays

IS-A

- IS-A Relationship
  - Human IS-A Mammal
  - Salmon IS-A Fish
  - ACartesianPoint IS-A Point

T1 **IS-A** T2 if T1 has all traits (methods and variables in Java) of T2

IS-A Relationship

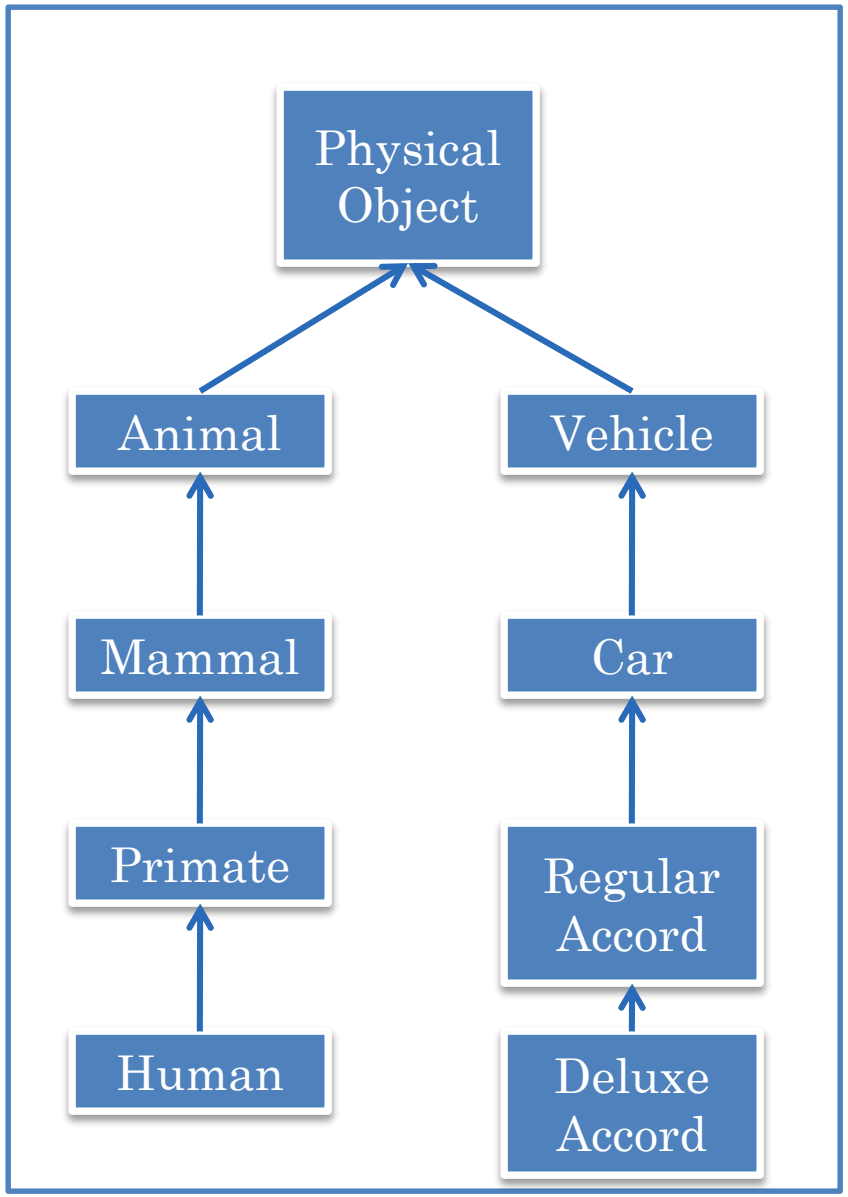


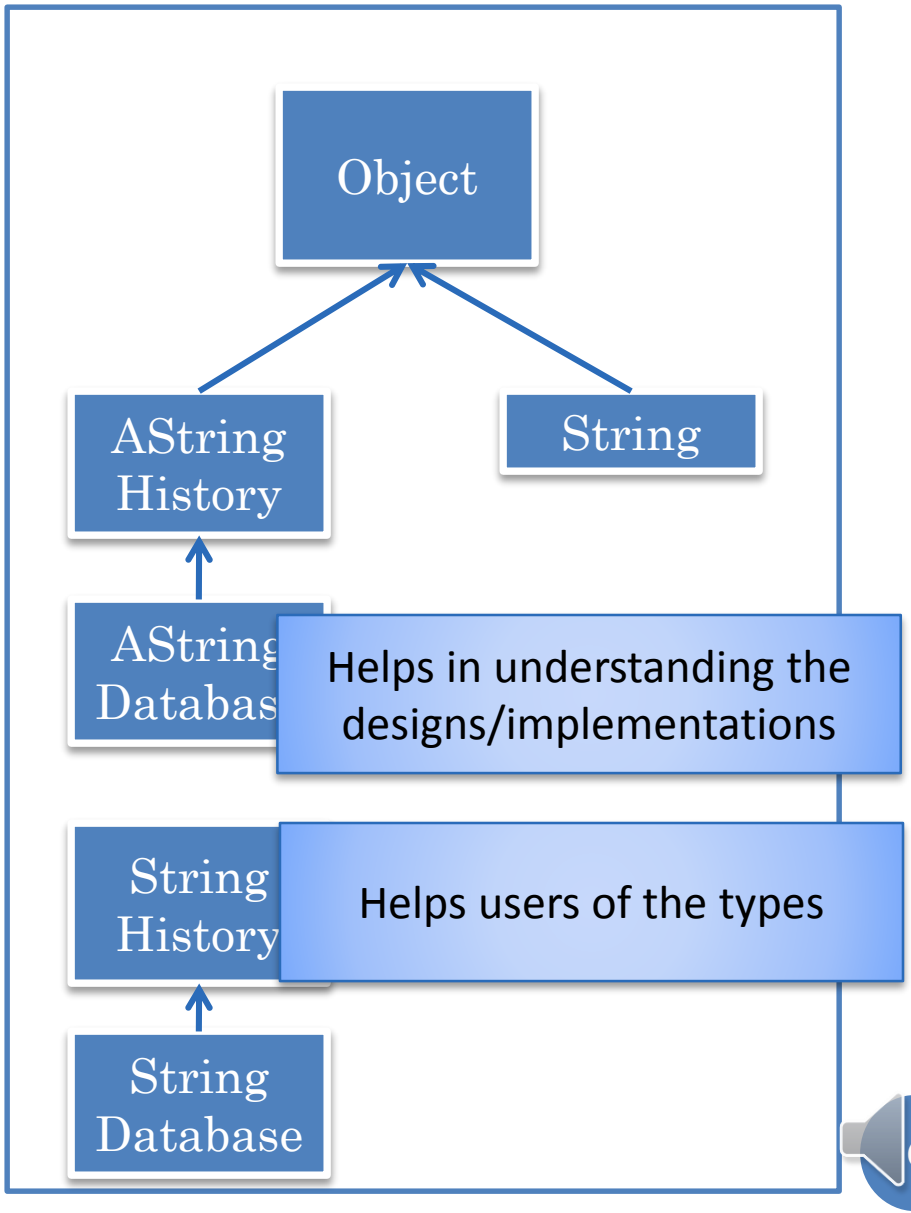
IS-A Definition

- Implements: T1 implements T2 => T1 IS-A T2
  - AStringHistory IS-A StringHistory
  - AStringDatabase IS-A StringDatabase
- Extends: T1 extends T2 => T1 IS-A T2
  - StringDatabse IS-A StringHistory
  - AStringDatabse IS-A AStringHistory
- Transitive:
  - T1 IS-A T2
  - T2 IS-A T3
  - => T1 IS-A T3
    - AStringDatabase IS-A StringHistory
- Reflexive:
  - T1 == T2 => T1 IS-A T2

    StringHistory IS-A StringHistory

Why Interesting?





Exact Typing and Overloaded Methods

```
public void print (ABMISpreadsheet aBMISpreadsheet) {
  System.out.println ("Height:" + aBMISpreadsheet.getHeight());
  System.out.println ("Weight:" + aBMISpreadsheet.getWeight());
  System.out.println("BMI:" + aBMISpreadsheet.getBMI());
}
```

```
print (new ABMISpreadsheet());
```

```
public void print (AnotherBMISpreadsheet aBMISpreadsheet) {
  System.out.println ("Height:" + aBMISpreadsheet.getHeight());
  System.out.println ("Weight:" + aBMISpreadsheet.getWeight());
  System.out.println("BMI:" + aBMISpreadsheet.getBMI());
}
```

```
print (new AnotherBMISpreadsheet());
```

Flexible Typing and Polymorphism

```
public void print (BMISpreadsheet aBMISpreadsheet) {
  System.out.println ("Height:" + aBMISpreadsheet.getHeight());
  System.out.println ("Weight:" + aBMISpreadsheet.getWeight());
  System.out.println("BMI:" + aBMISpreadsheet.getBMI());
}
```

```
print (new AnotherBMISpreadsheet());
print (new ABMISpreadsheet());
```

ABMISpreadsheet **implements** BMISpreadsheet

ABMISpreadsheet **IS-A** BMISpreadsheet

AnotherBMISpreadsheet **implements** BMISpreadsheet

AnotherBMISpreadsheet **IS-A** BMISpreadsheet

T1 **IS-A** T2 if T1 has all traits (methods and variables in Java) of T2

IS-A (Review)

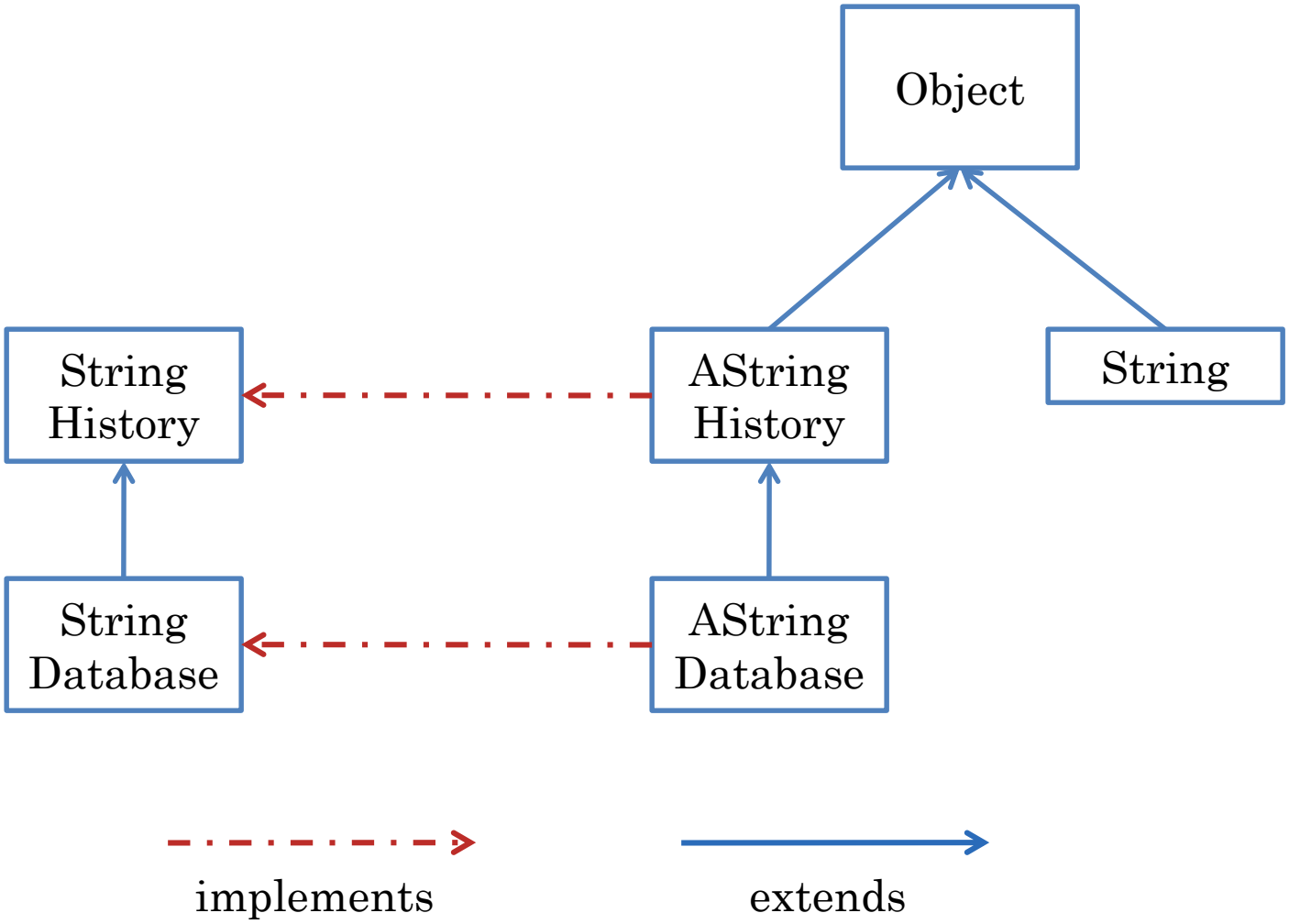
- IS-A Relationship
  - Human IS-A Mammal
  - Salmon IS-A Fish
  - ACartesianPoint IS-A Point

T^1 **IS-A** T^2 if T^1 has all traits (methods and variables in Java) of T^2

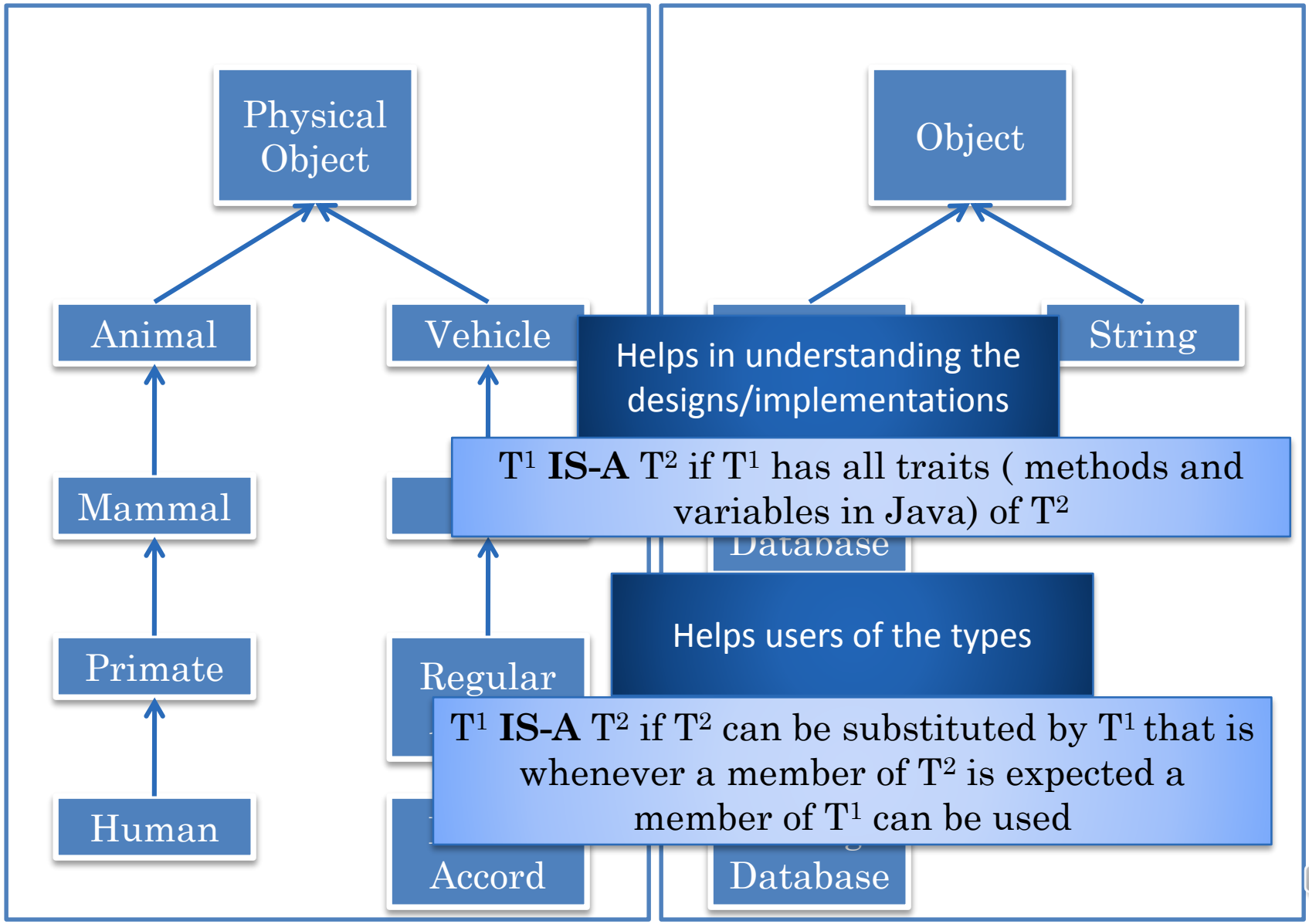
T^1 **IS-A** T^2 if the set of members of T^1 is a subset of the set of members of T^2

T^1 **IS-A** T^2 if T^2 can be substituted by T^1 that is whenever a member of T^2 is expected a member of T^1 can be used

IS-A Relationship (Review)



Why Interesting? (Review)



Flexible Typing and Polymorphism (Review)

```
public void print (BMISpreadsheet aBMISpreadsheet) {
  System.out.println ("Height:" + aBMISpreadsheet.getHeight());
  System.out.println ("Weight:" + aBMISpreadsheet.getWeight());
  System.out.println("BMI:" + aBMISpreadsheet.getBMI());
}
```

```
print (new AnotherBMISpreadsheet());
print (new ABMISpreadsheet());
```

ABMISpreadsheet **implements** BMISpreadsheet

ABMISpreadsheet **IS-A** BMISpreadsheet

AnotherBMISpreadsheet **implements** BMISpreadsheet

AnotherBMISpreadsheet **IS-A** BMISpreadsheet

T^1 **IS-A** T^2 if T^2 can be substituted by T^1 that is whenever a member of T^2 is expected a member of T^1 can be used

Type-Checking Examples

```
AStringHistory stringHistory = new AStringHistory();
```

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

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

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

Value of type T1 can be assigned to variable of type T2

If T1 IS-A T2

Rationale and Casting

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

stringDatabase.clear()

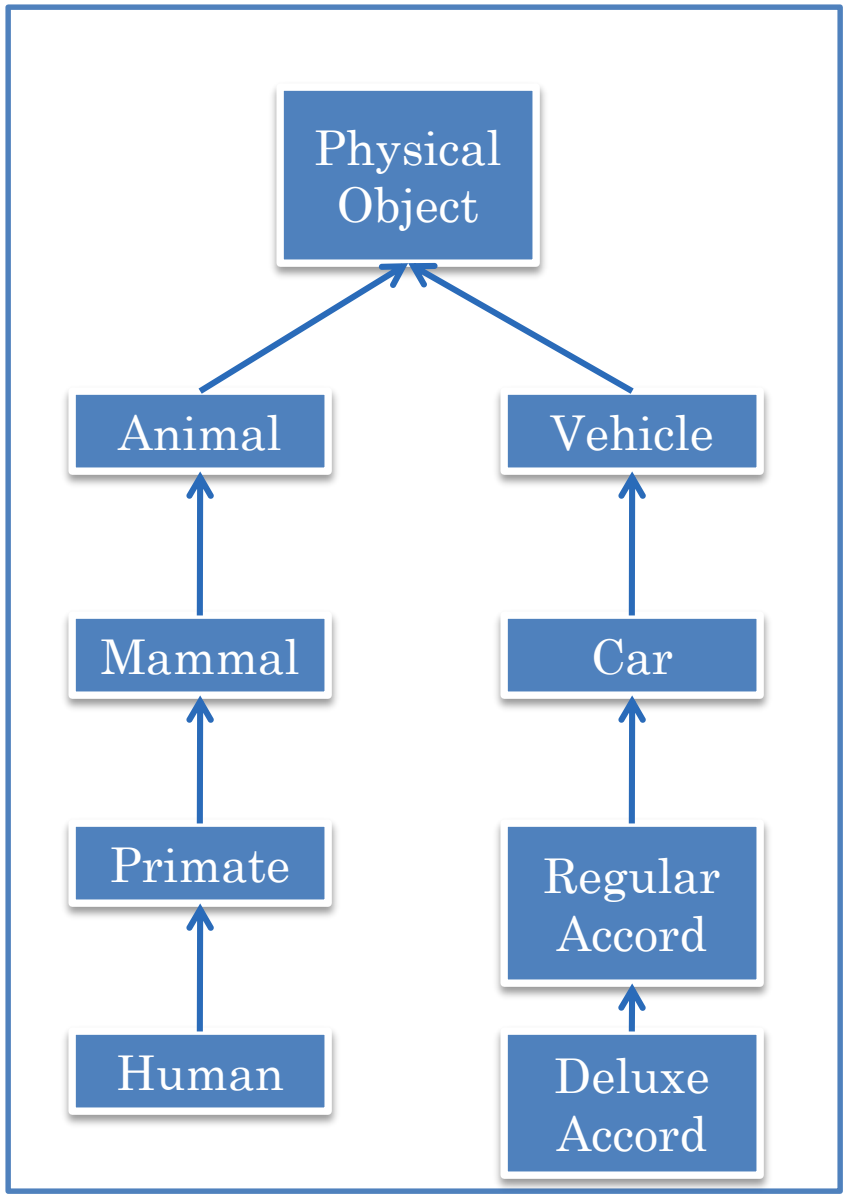
StringHistory stringHistory = **new** AStringDatabase();

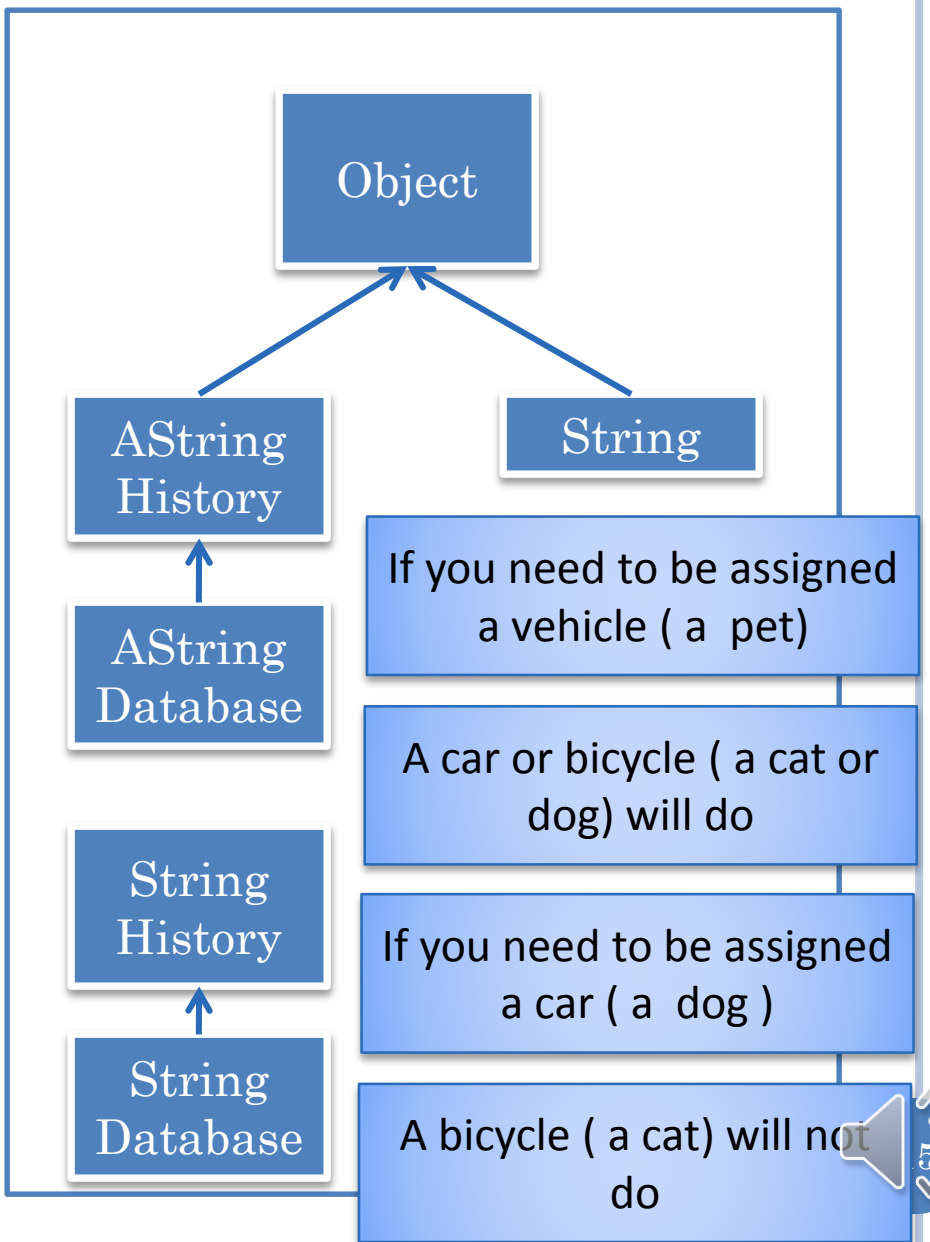
stringHistory.size()

stringHistory .clear()

((StringDatabase) stringHistory) .clear()

Analogy for Type Checking Rules?



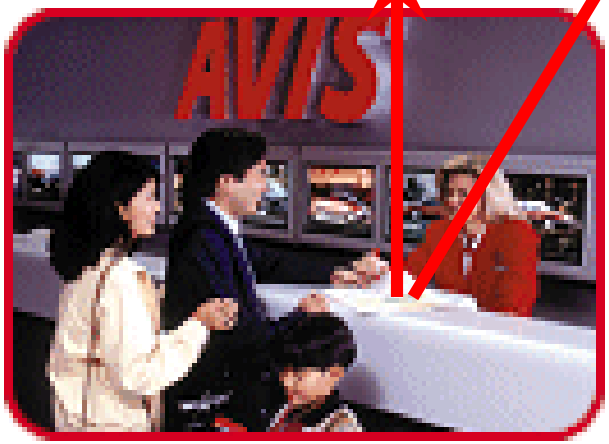


Getting an Upgrade

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

Regular Model Requested

Deluxe Model Assigned



myCar.steer();

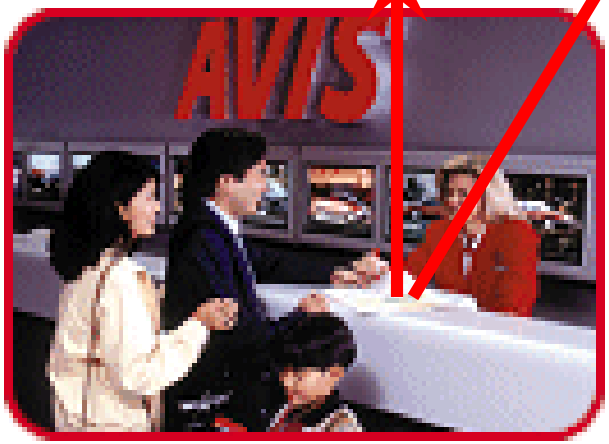
myCar.
setCity(“Raleigh”);

((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

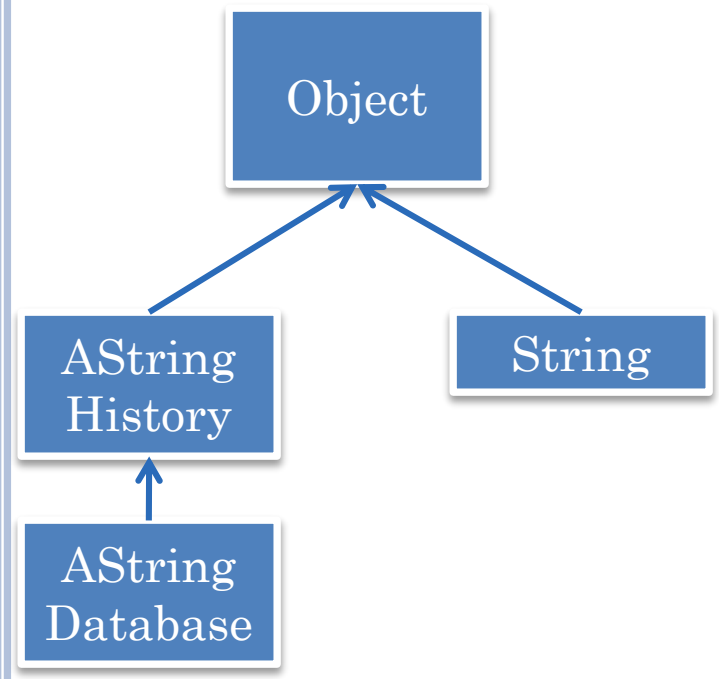
IS-A Rules Revisited

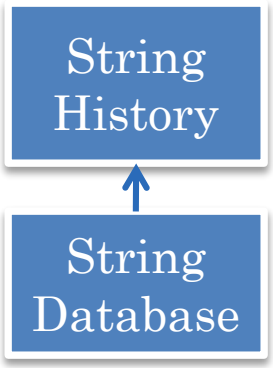
- Extends: T1 extends T2 => T1 IS-A T2
- Implements: T1 implements T2 => T1 IS-A T2
- Transitive:
  - T1 IS-A T2
  - T2 IS-A T3
  - => T1 IS-A T3
- Reflexive:
  - T1 == T2 => T1 IS-A T2

Value of type T1 can be assigned to variable of type T2, if T1 is a T2

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

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;
```

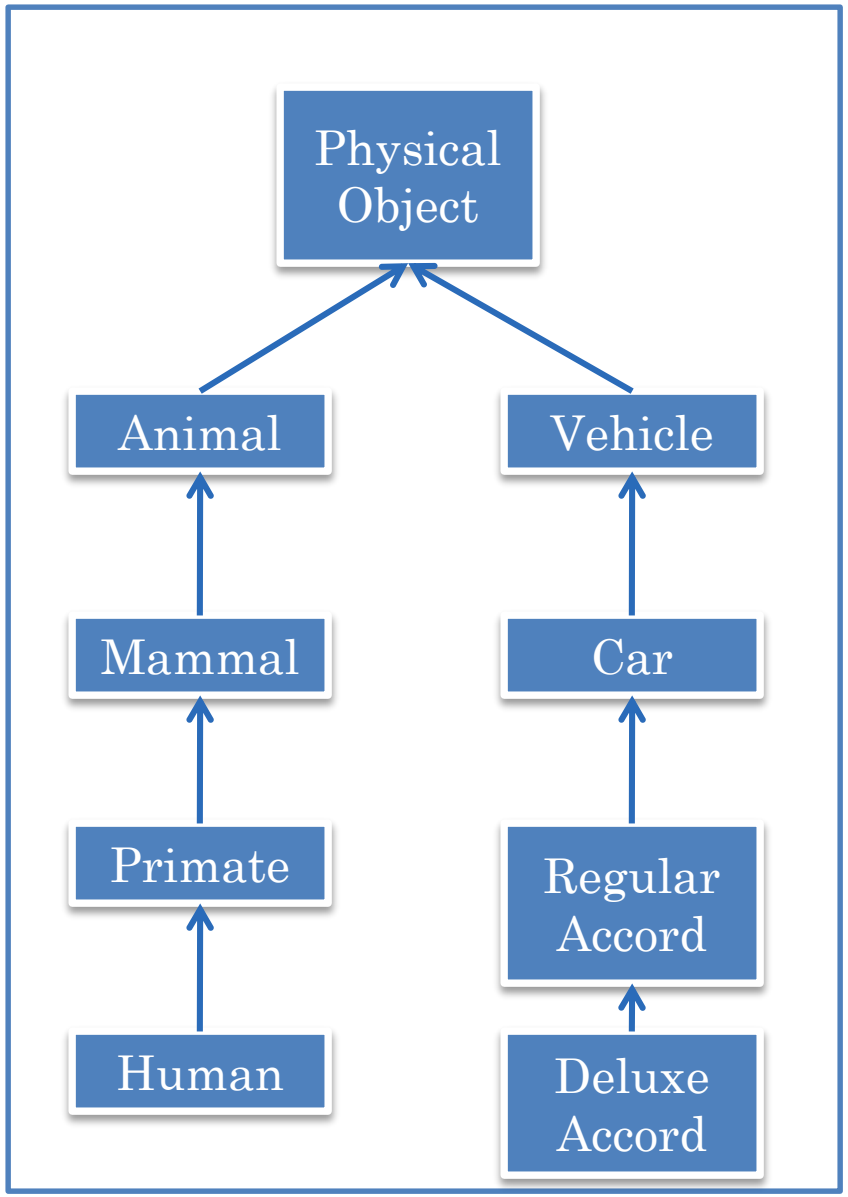
```
double d = 2;
```

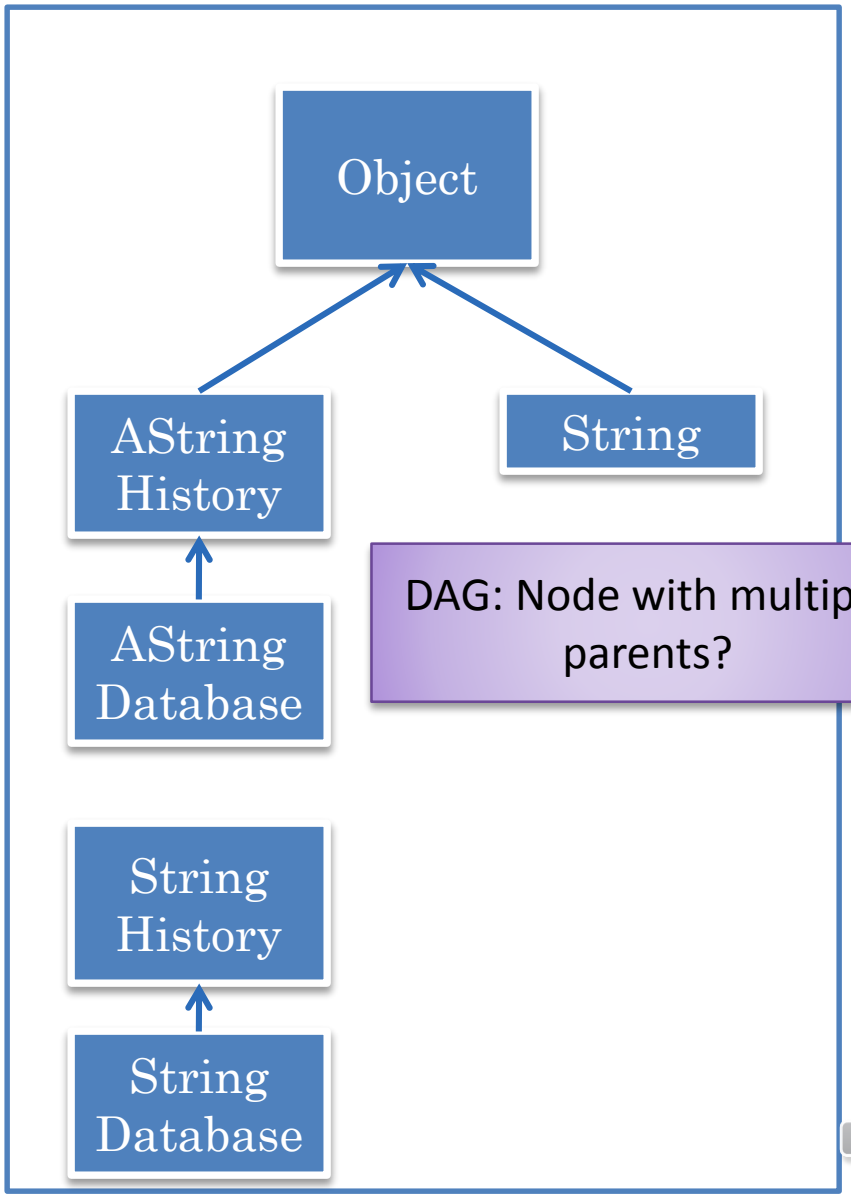
```
int i=  (int) 2.5;
```

Assignment Rules for Primitive Types

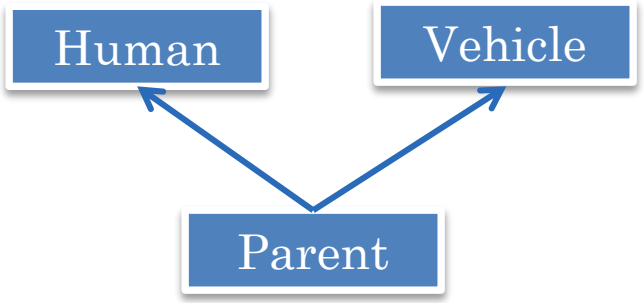
- If T1 narrower than T2 (Set of instances of T1 $\subseteq$ Set of instances of T2)
- Expression of type T1 can be assigned to variable of type T2
- Expression of type T2 can be assigned to variable of type T1 with cast
  - Primitive casts can convert values
    - **(int)** double converts double to int value
  - Object casts never convert values!

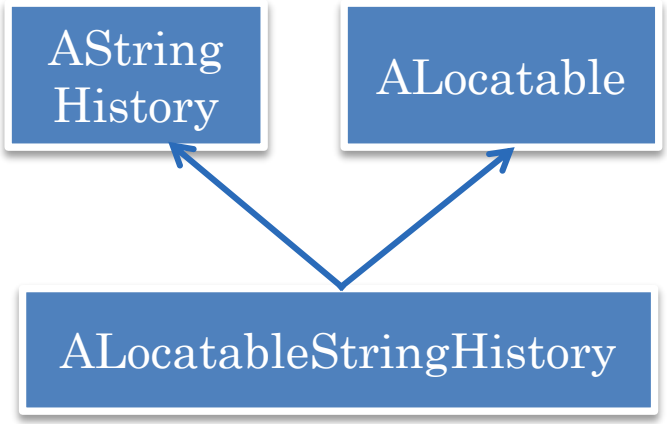
Tree


Physical Object
Animal
Vehicle
Mammal
Car
Primate
Regular Accord
Human
Deluxe Accord


Object
AString History
String
AString Database
DAG: Node with multiple parents?
String History
String Database

DAG? Multiple Inheritance?

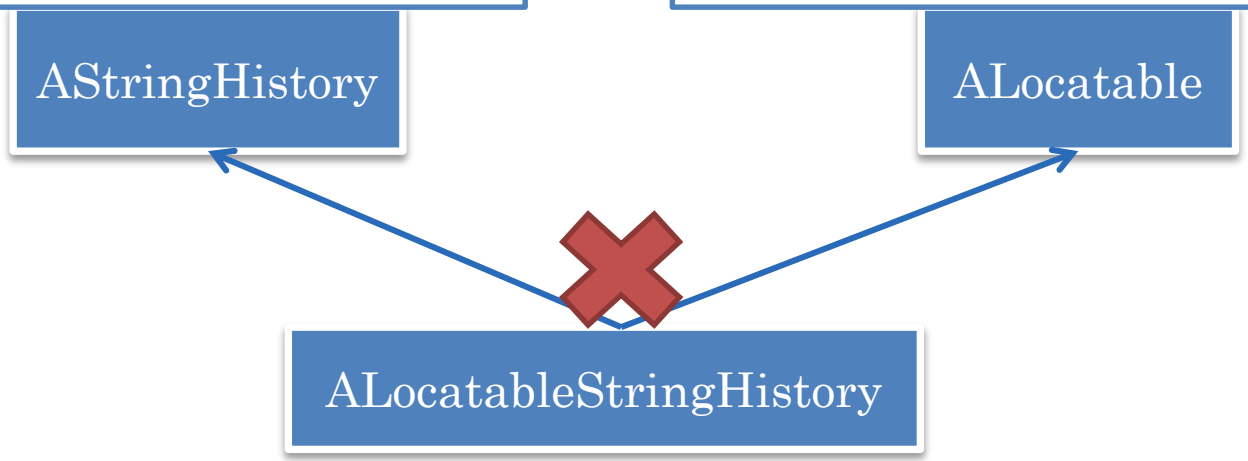




Problem with Multiple Class Inheritance

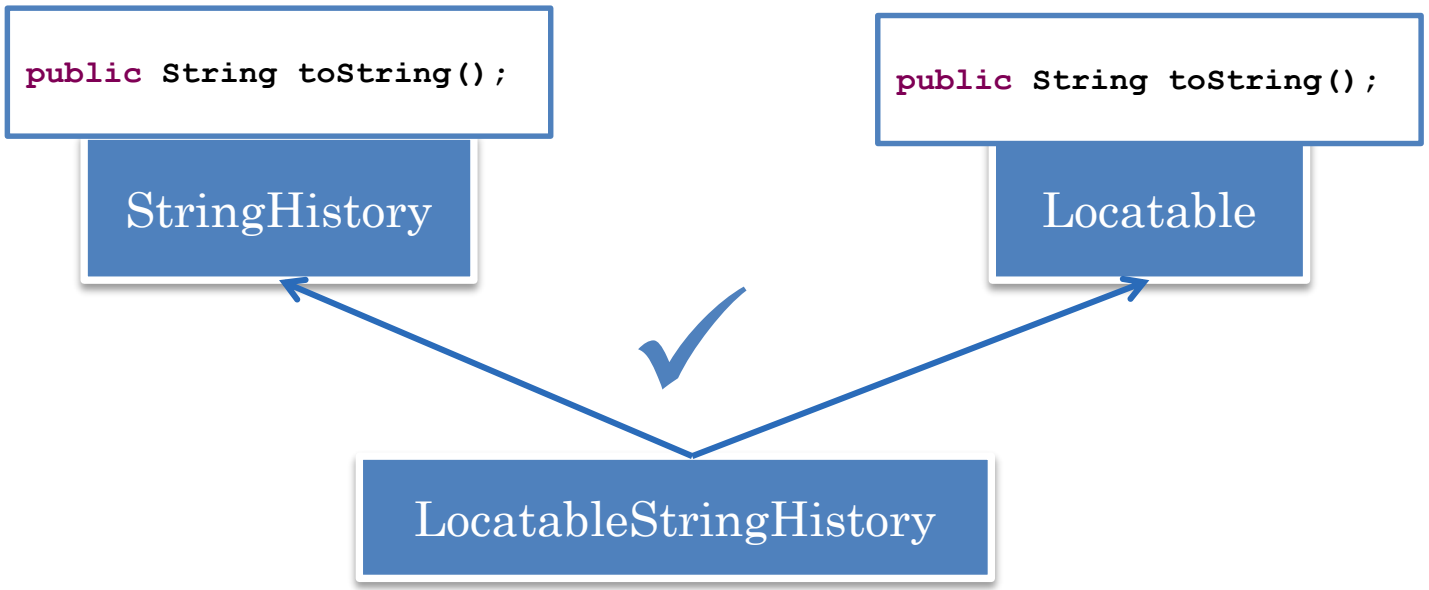
```
public String toString() {
  String retVal = "";
  for (int i = 0; i < size; i++) {
    String separator =
       (i == size- 1)?"":":";
    retVal += separator + contents[i];
  }
  return retVal;
}
```

```
public String toString() {
   return  x + "," + y;
}
```



Which toString() should be called? They are different methods with the same header.

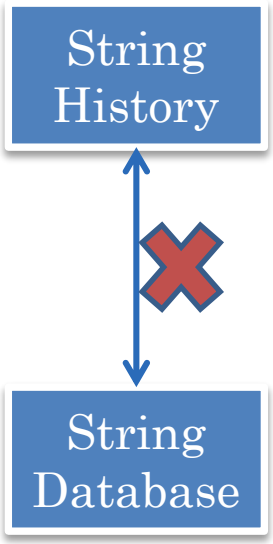
Problem with Multiple Interface Inheritance?



Duplicates added to a set of methods

Inheriting the fact you have eyes vs. a particular kind of eyes

Cycles?



Conceptually If a type is a special case of another type, it cannot be a general case of that type unless it is equal to the other type

Implementing Multiple Interfaces

```
public class ABMISpreadsheetAndCalculator implements BMISpreadsheet, BMICalculator{
    double height, weight, bmi;
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
        bmi = calculateBMI(height, weight);
    }
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
        bmi = calculateBMI(height, weight);
    }
    public double getBMI() {
        return bmi;
    }
    public double calculateBMI(double height, double weight) {
        return weight/(height*height);
}
```

Typing Issue

```
public static void main (String[] args) {
    BMISpreadsheet bmiSpreadsheet = new ABMISpreadsheetAndCalculator();
    bmi = bmiSpreadsheet.getBMI();
// bmi = bmiSpreadsheet.calculateBMI(1.77, 75);
}
```

Multiple Interface Inheritance

```
public interface BMISpreadsheetAndCalculator extends BMISpreadsheet, BMICalculator{

}
```

An interface can extend multiple interfaces

A class cannot extend multiple classes (in Java)

An extended interface or class may not have extra methods!

Implementing Single Interface

```
public class ABMISpreadsheetAndCalculator implements BMISpreadsheetAndCalculator{
    double height, weight, bmi;
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
        bmi = calculateBMI(height, weight);
    }
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
        bmi = calculateBMI(height, weight);
    }
    public double getBMI() {
        return bmi;
    }
    public double calculateBMI(double height, double weight) {
        return weight/(height*height);
}
```

Typing

```
public static void main (String[] args) {
   BMISpreadsheet bmiSpreadsheet = new ABMISpreadsheetAndCalculator();
   bmi = bmiSpreadsheet.getBMI();
// bmi = bmiSpreadsheet.calculateBMI(1.77, 75);
}
```

```
public static void main (String[] args) {
   BMISpreadsheetAndCalculator bmiSpreadsheet =
      new ABMISpreadsheetAndCalculator();
   double bmi = bmiSpreadsheet.calculateBMI(1.77, 75);
   bmi = bmiSpreadsheet.getBMI();
}
```

Every public method of a class must be in an interface

A class should implement a single (possibly multiply extended) interface

Inheriting Constants

```
public class AStringHistory implements StringHistory {
        public final int MAX_SIZE = 50;
        ….
}
```

```
public class AStringDatabase
                extends AStringHistory implements StringDatabase {
  public AStringDatabase() {
      System.out.println(MAX_SIZE);
  }
   ….
}
```

Ambiguous.

```
public interface StringDatabase  extends StringHistory  {
          public final int MAX_SIZE = 20;
   ….
}
```

Inheriting Constants

```
public class AStringHistory implements StringHistory {
        public final int MAX_SIZE = 50;
        ….
}
```

```
public class AStringDatabase
                extends AStringHistory implements StringDatabase {
….
}
```

```
public interface StringDatabase  extends StringHistory  {
          public final int MAX_SIZE = 20;
    ….
}
```

```
StringDatabase stringDatabase = new AStringDatabase();
System.out.println(stringDatabase.MAX_SIZE);
```

Interface constant used

Inheriting Constants

```
public class AStringHistory implements StringHistory {
        public final int MAX_SIZE = 50;
        ….
}
```

```
public class AStringDatabase
                extends AStringHistory implements StringDatabase {
….
}
```

```
public interface StringDatabase  extends StringHistory  {
        public final int MAX_SIZE = 20;
    ….
}
```

```
AStringDatabase stringDatabase = new AStringDatabase();
System.out.println(stringDatabase.MAX_SIZE);
```

Ambiguous

ObjectEditor and Inheritance

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

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

```
public class AMutablePoint extends ACartesianPoint
    ts Point {…}
```

Ambiguous

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

Class of the object gets precedence over its super types

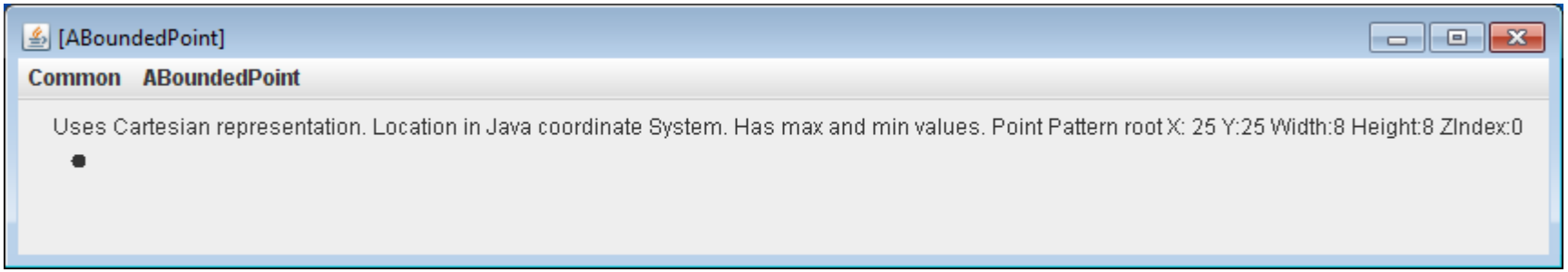
Java does not define inheritable annotations, this is ObjectEditor simulating inheritance using its own rules

Multiple Inheritance in ObjectEditor

```
@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 AMuatablePoint
implements BoundedPoint {…}
```



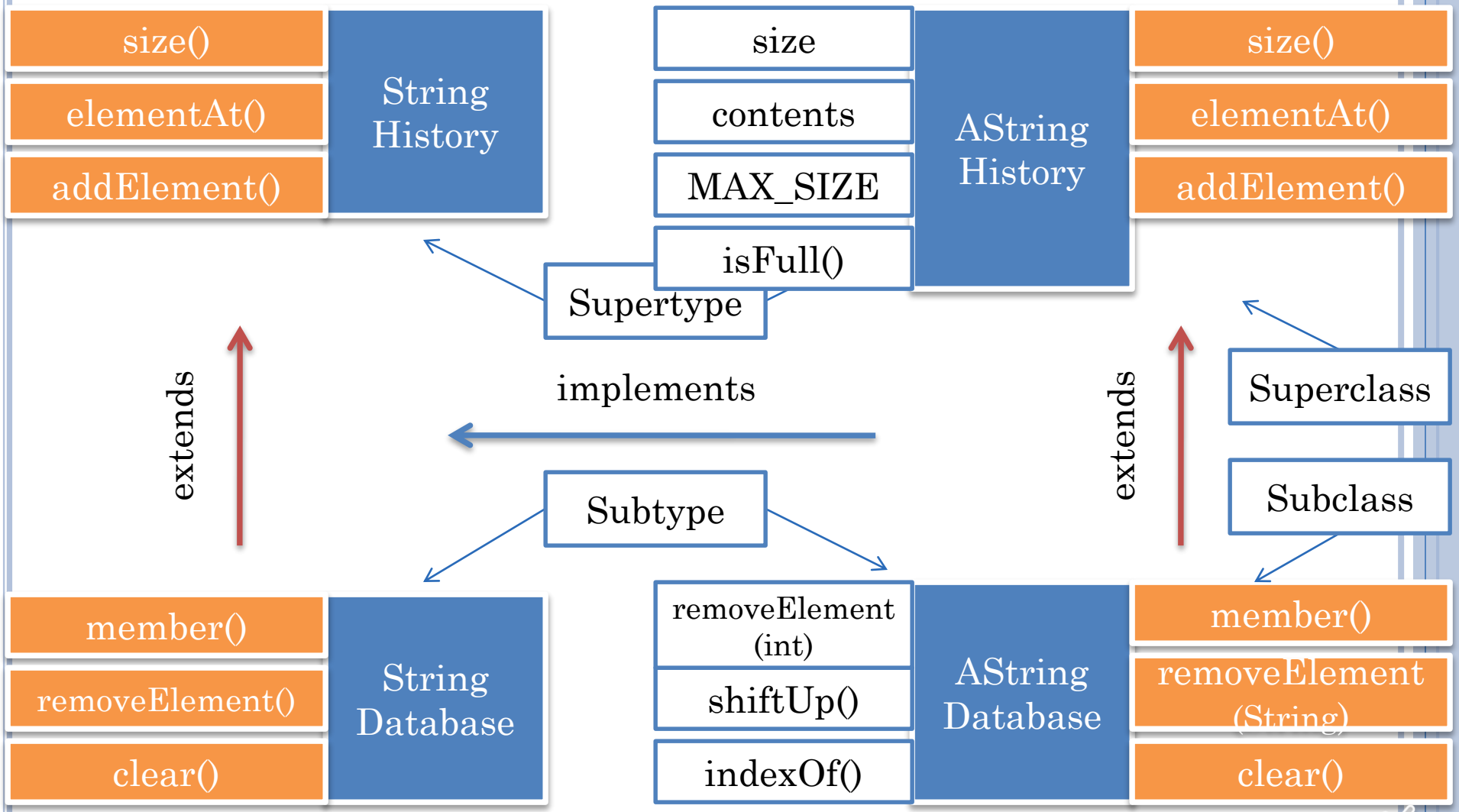
Sometimes inherited attributes can be combined rather than overridden.

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

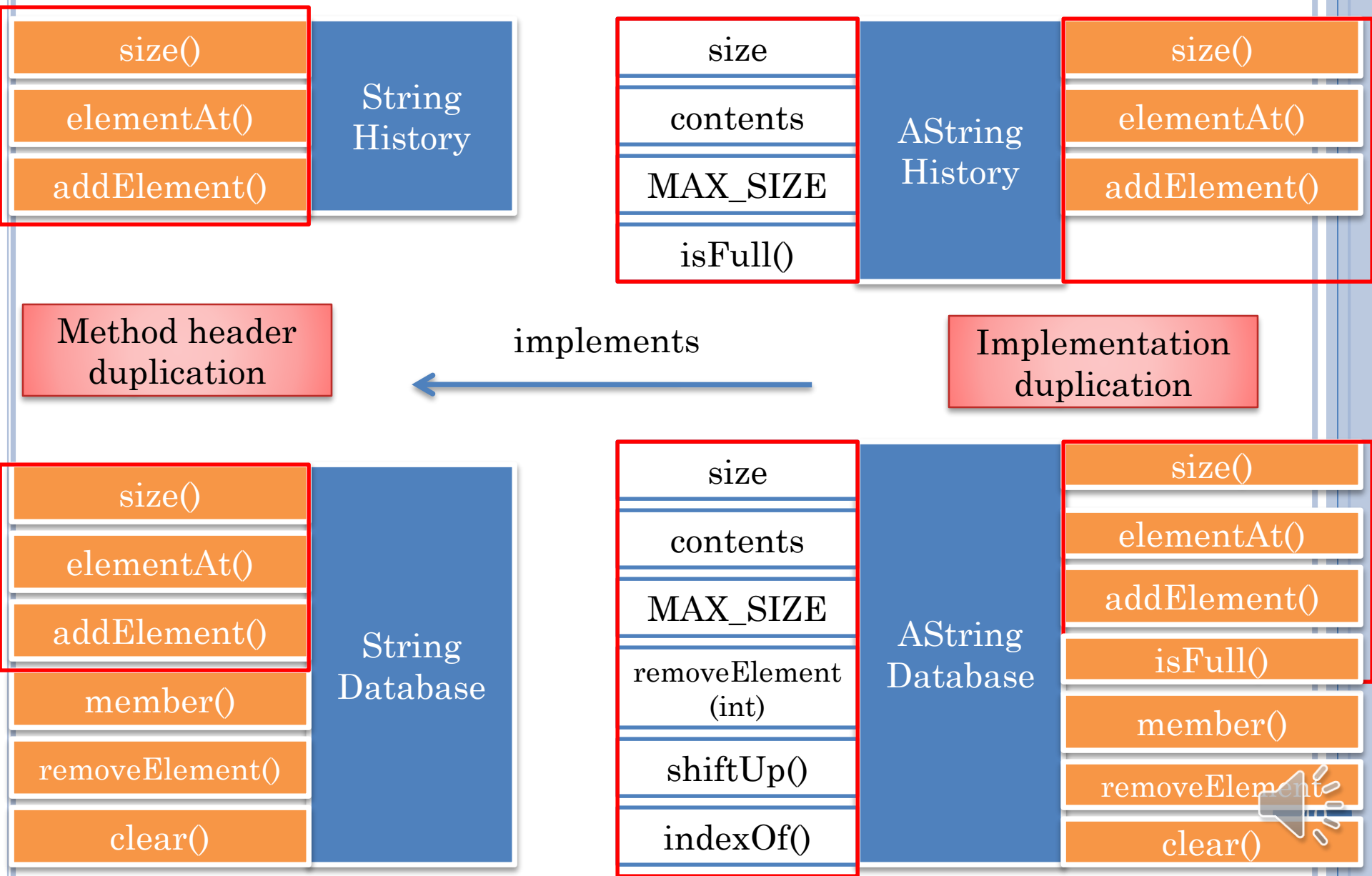
How to get Inheritance Right?

If type T1 contains a super set of the method headers in another type T2, then T1 should extend T2?

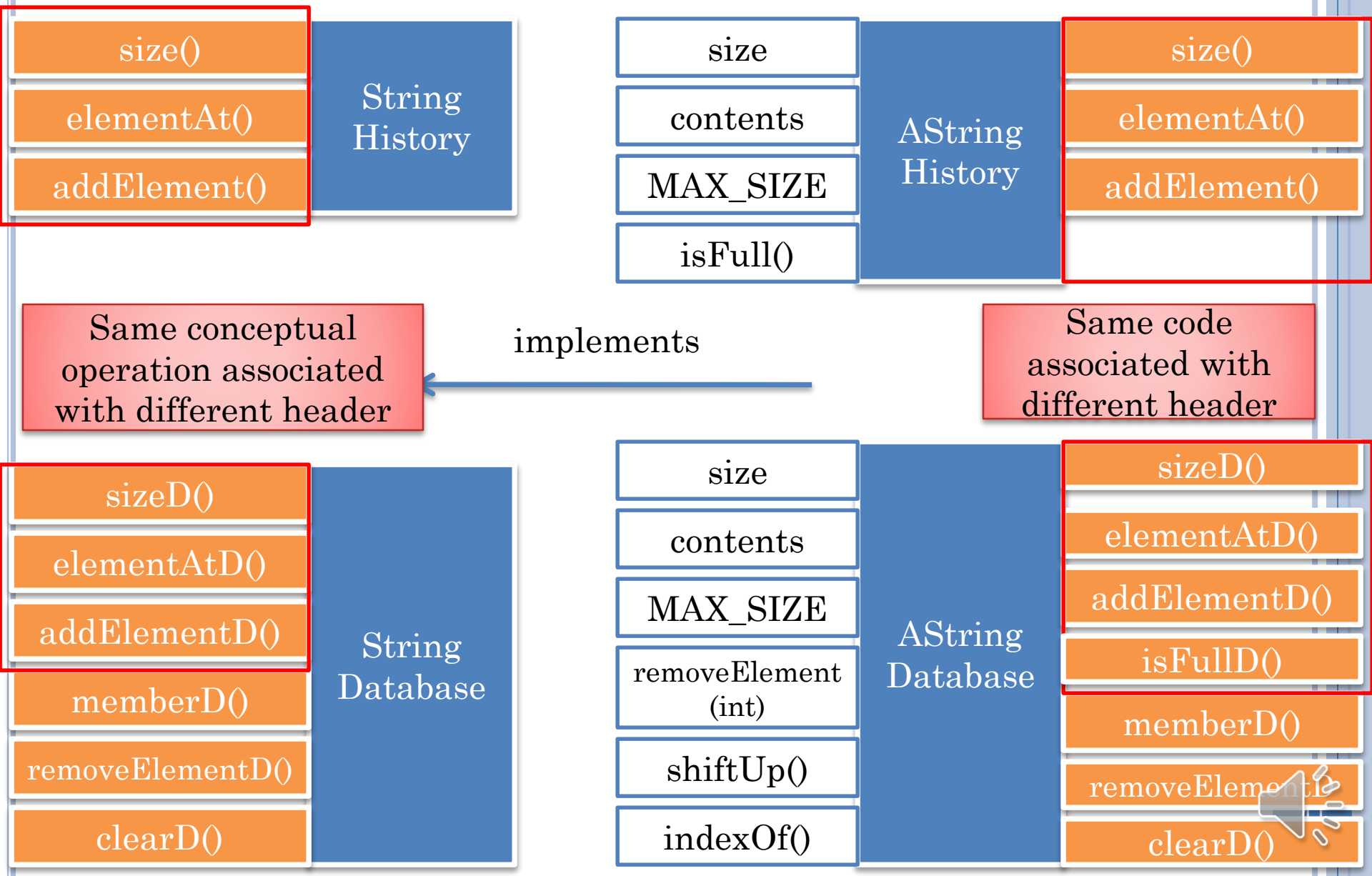
Correct Use of Inheritance



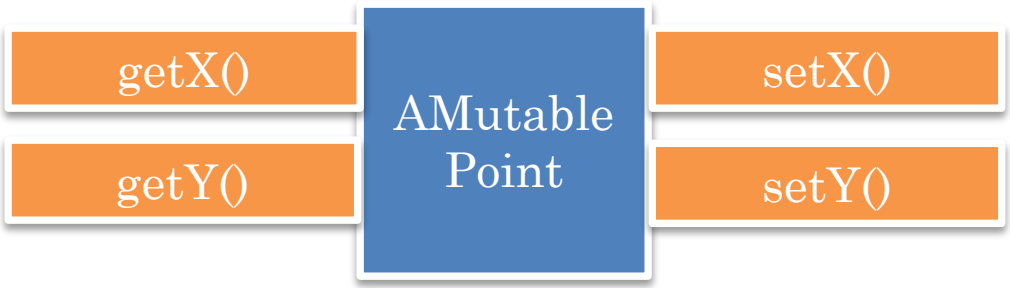
Not Using Inheritance When you can

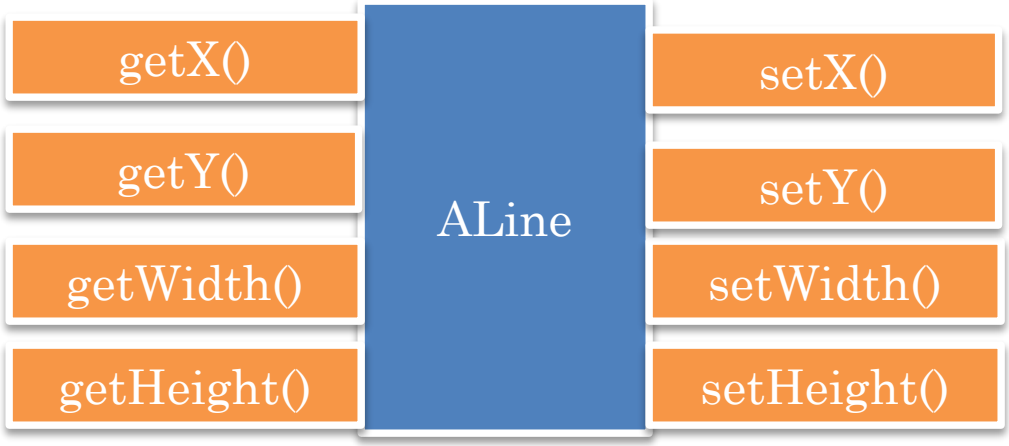


Not Using Inheritance When you can?



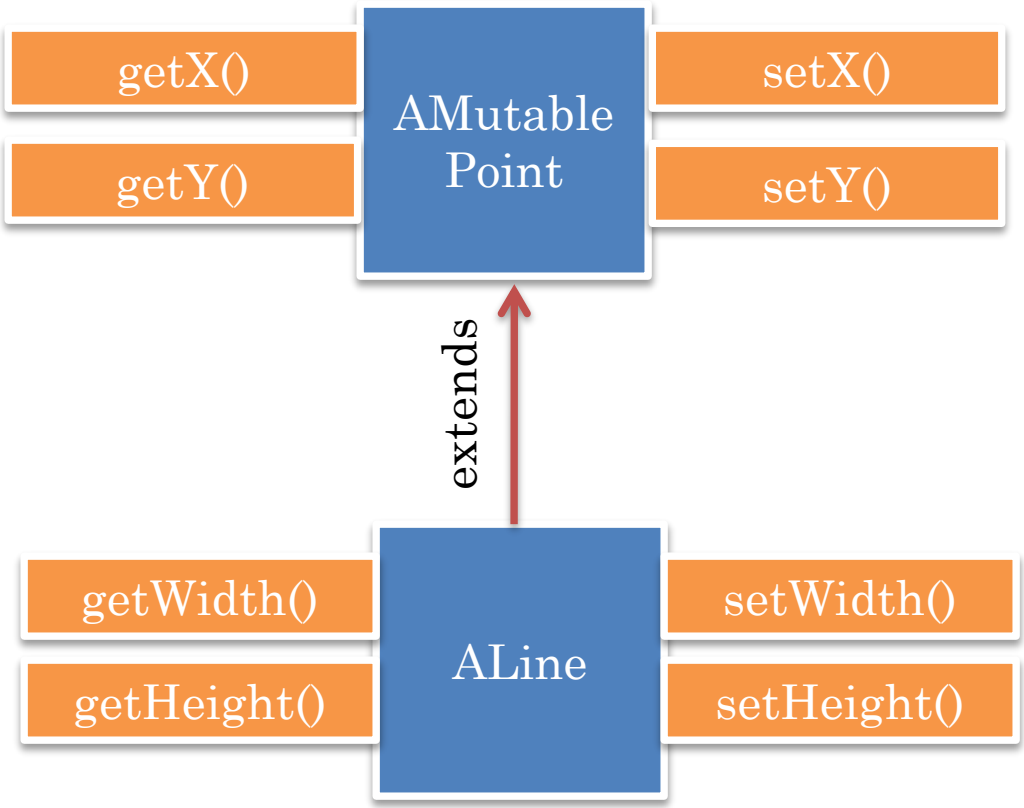
Point and Line





Inherit?

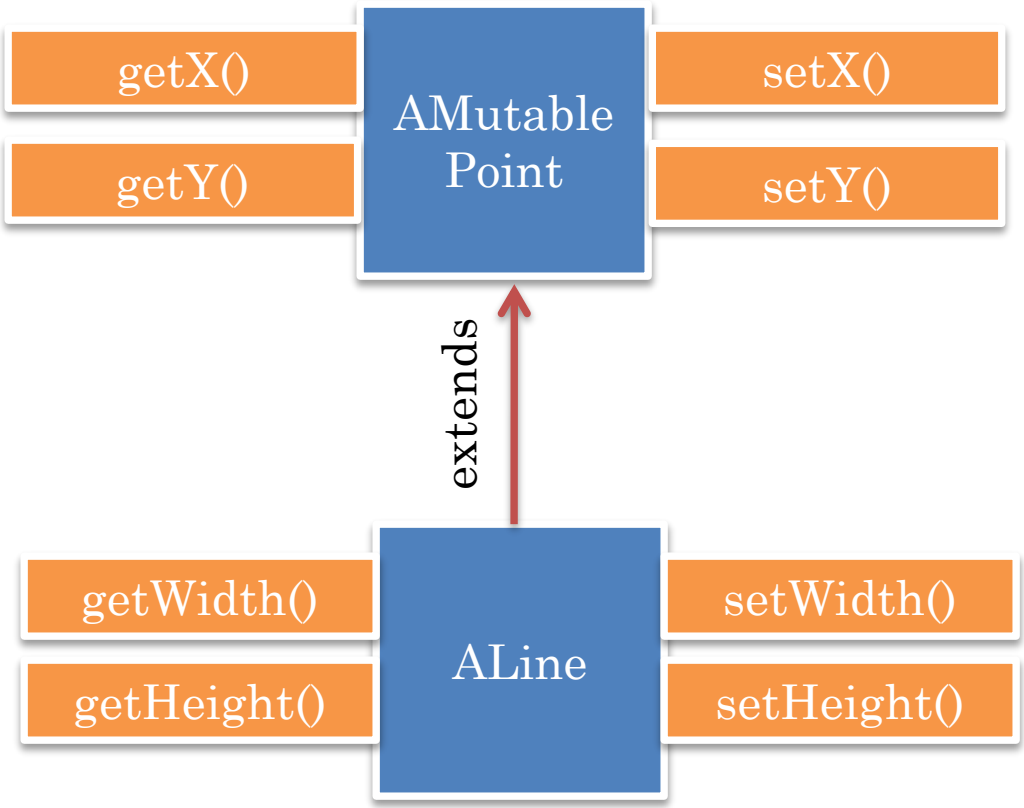
A Line is not a Point!



T^1 **IS-A** T^2 if T^2 can be substituted by T^1 that is whenever a member of T^2 is expected a member of T^1 can be used

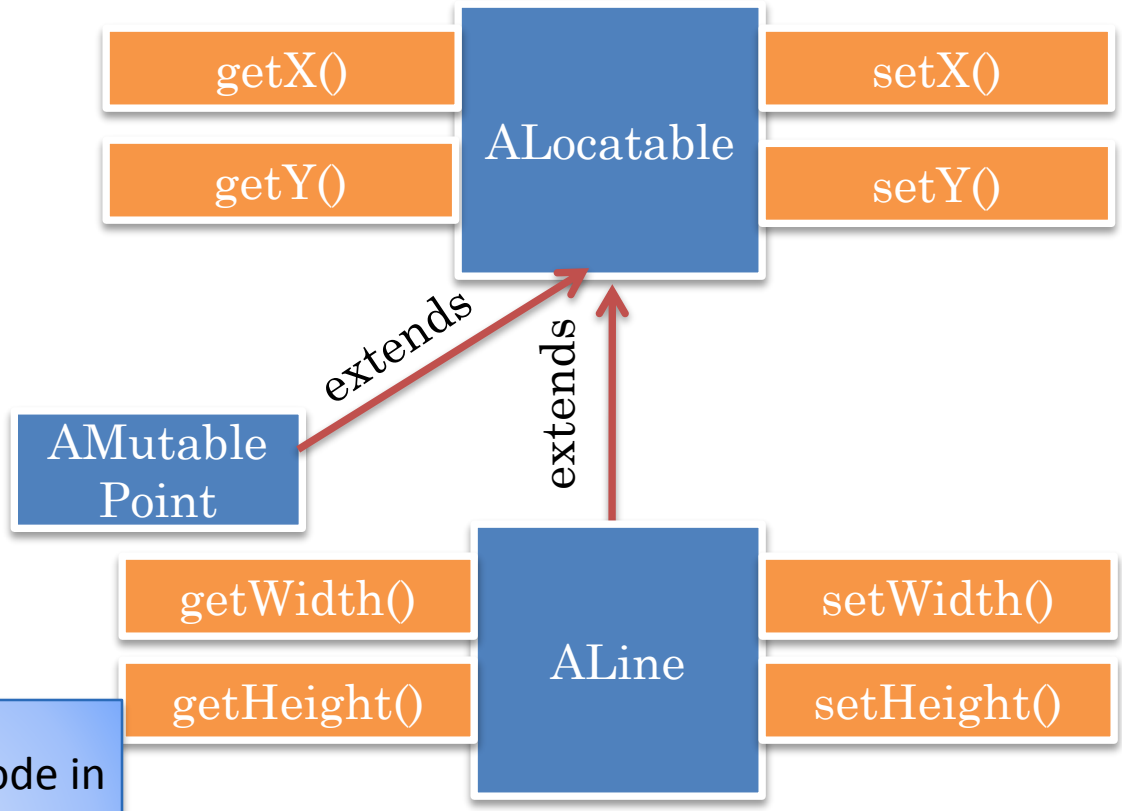
Inherit? (Review)

A Line is not a Point!



T^1 **IS-A** T^2 if T^2 can be substituted by T^1 that is whenever a member of T^2 is expected a member of T^1 can be used

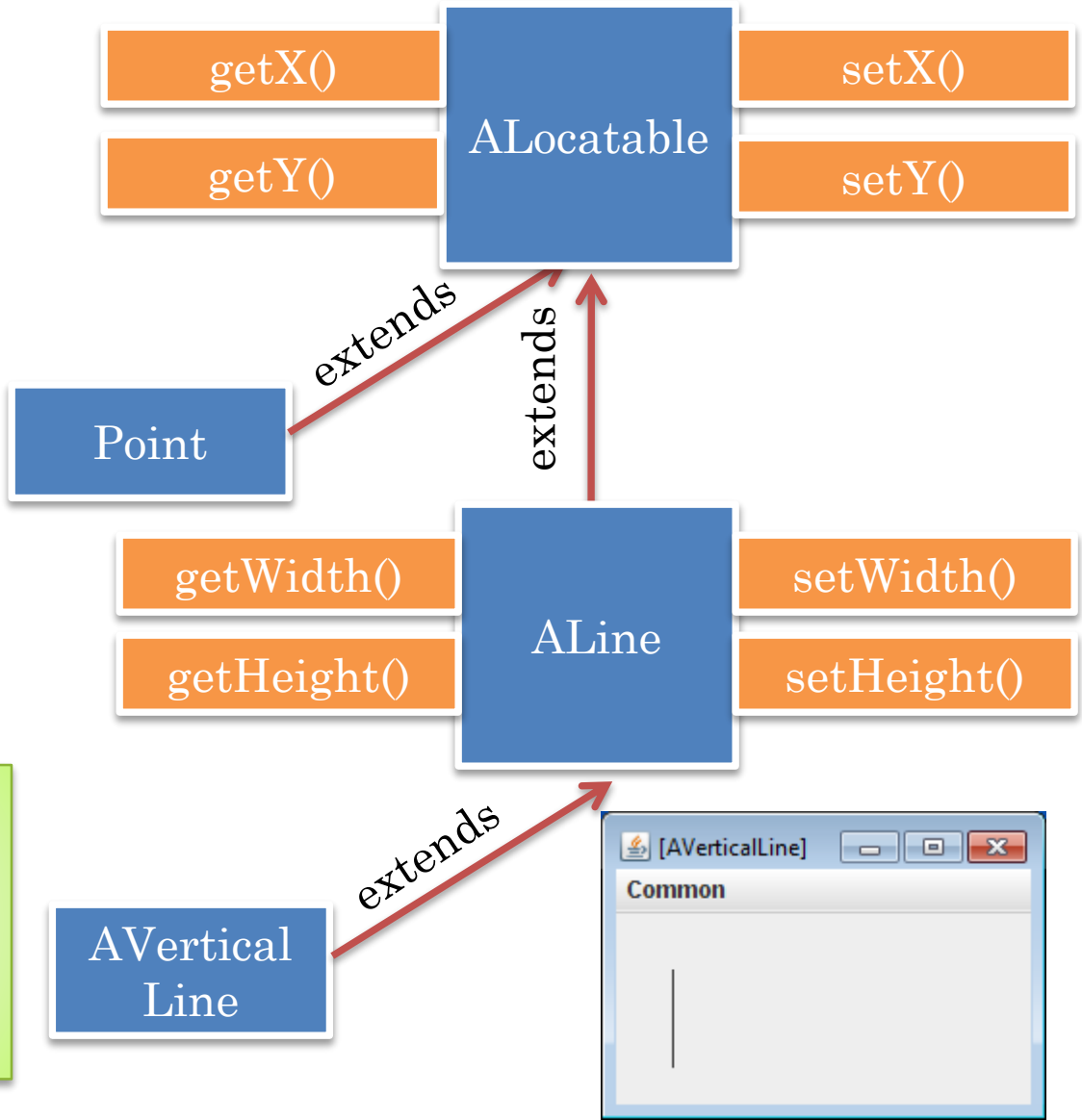
Correct Use



An extending interface or method may not add extra methods!

Often, to share code in two existing types, a common new super-type is needed

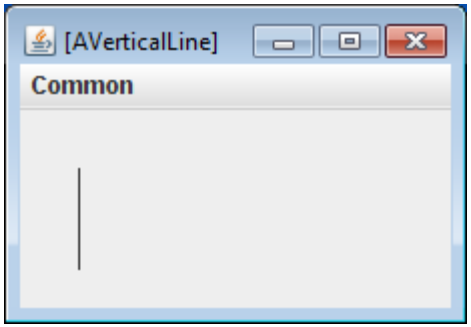
AVERTICALLINE



T^1 **IS-A** T^2 if the set of members of T^1 is a subset of the set of members of T^2

Overriding Code and Limitations

Subclass constructor can have fewer parameters



Cannot disable inherited methods in Java

```
public class AVerticalLine extends ALine {
  public AVerticalLine (int initX, int initY, int initHeight) {
    super (initX, initY, 0, initHeight);
  }
  public void setWidth(int newVal) {

  }
}
```

Setter?

```
public static boolean checkLine(Line aLine, int aWidth
  aLine.setWidth(aWidth);
  return aLine.getWidth() == aWidth;
}
```

Some checks written for Aline do not work for AVerticalLine

```
System.out.println(checkLine (new ALine(10, 10, 50, 50), 10));
System.out.println(checkLine (new AVerticalLine(10, 10, 50), 10));
```

→ true

→ false

Two Potential Principle Violations

Getter/Setter Rule

getP() == v if previous statement is setP(v)

Liskov Substitution Principle

A subtype passes all the checks passed by its supertype

→ Very limited (efficiency based) overriding

By this definition a Set is not a Database

Principles Followed and Not Followed

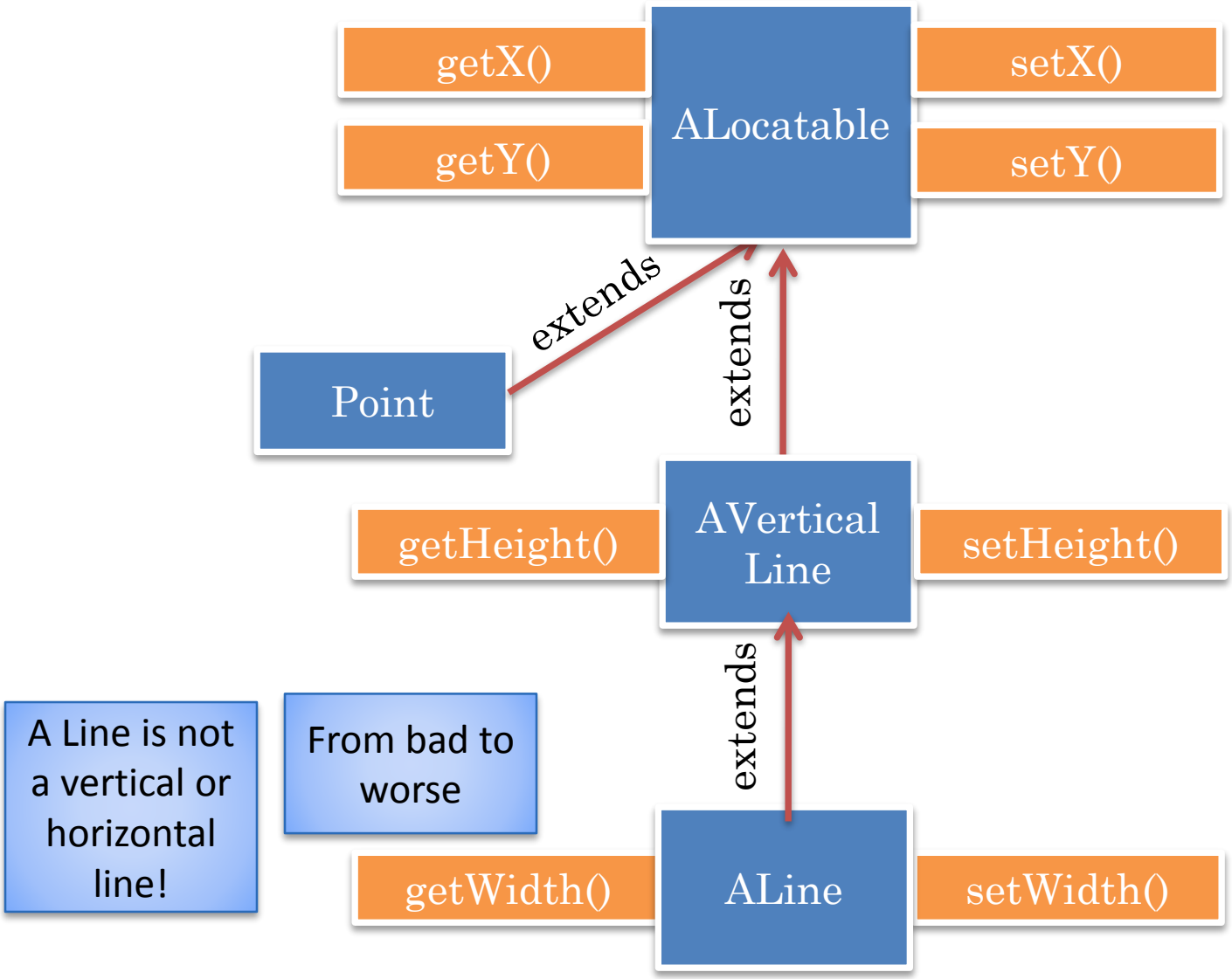
T^1 **IS-A** T^2 if the set of members of T^1 is a subset of the set of members of T^2

getP() == v if previous statement is setP(v)

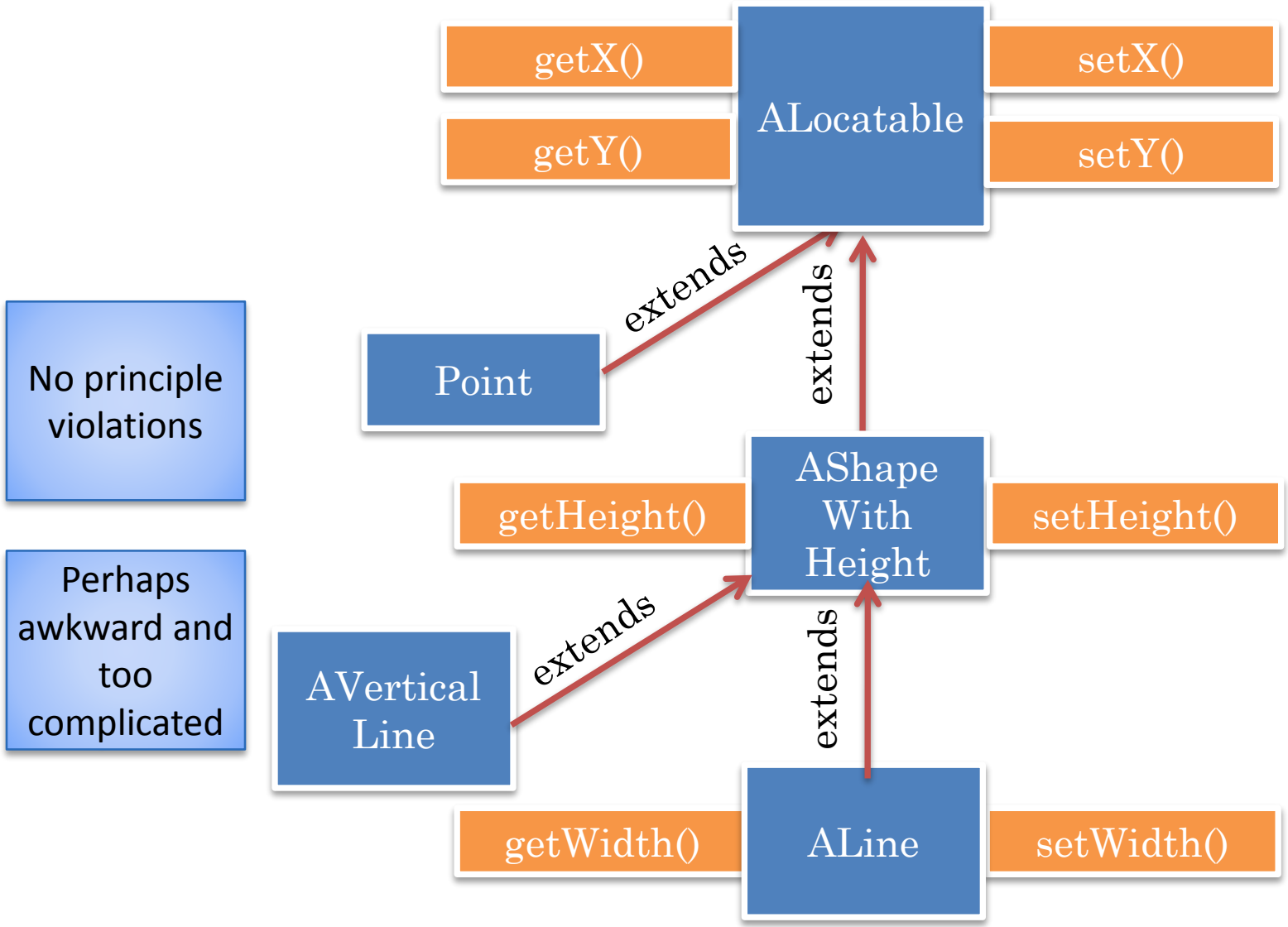
A subtype passes all the checks passed by its supertype

Confusing and the solution depends on you!

Refactor?

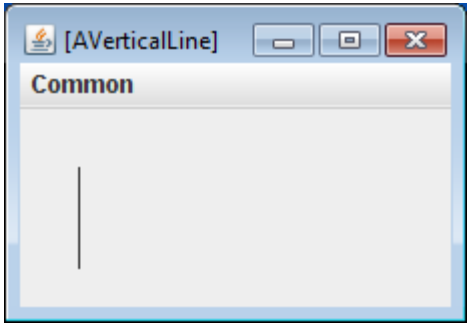

getX()
getY()
ALocatable
setX()
setY()
extends
Point
extends
getHeight()
AVertical Line
setHeight()
extends
A Line is not a vertical or horizontal line!
From bad to worse
getWidth()
ALine
setWidth()

Refactor?


getX()
getY()
ALocatable
setX()
setY()
extends
extends
Point
AShape With Height
getHeight()
setHeight()
No principle violations
Perhaps awkward and too complicated
AVertical Line
extends
extends
getWidth()
ALine
setWidth()

Give an Error?

Check will fail but checker knows the failure is deliberate



We have made a “bug” a “feature”

```
public class AVerticalLine extends ALine {
  public AVerticalLine (int initX, int initY, int initHeight) {
    super (initX, initY, 0, initHeight);
  }
  public void setWidth(int newVal) {
        System.out.println(“Cannot change width”);
  }
}
```

Several Java collection classes follow this philosophy

Immutable collection classes are subclasses of mutable collection classes, throwing exceptions on read methods

Pure and Practical Answer

Delegation, which we will study later!

Inheritance Rules

- Ensure that same conceptual operations are associated with same method headers in different types (interfaces/classes)
- ObjectEditor patterns encourage reuse
- Use inheritance to get rid of method header/body duplication in different types (interfaces/classes)
- Inheritance should respect IS-A relationship
- Often delegation (we will see later) is the answer

Getting inheritance right is tricky!

Throw away your first implementation

Primitives and Objects

```
Object object = 5;
```

Primitive 5 converted into an instance of wrapper class Integer. Each primitive type is associated with a wrapper class that holds object versions of values of the primitive type. Integer IS-A object, so this should work.

```
int i = object
```

```
int i = (Integer) object
```

```
int i = ((Integer) object).intValue;
```

Value of wrapper class converted into primitive value

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

If program has an alternative plan then instanceof makes more sense

Alternatives

```
StringHistory stringHistory;
```

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

If program has an alternative plan then instanceof makes more sense

Usually considered bad programming to decide alternatives based on instanceof

INSTANCE OF: THE CASE AGAINST

```
StringHistory stringHistory;
```

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

Make the alternative actions implementations of the same method in the type of the variable used in instanceof

Declare display() in interface StringHistory and make different classes provide different implementations of it, which are chosen by Java (instead of user program) based on the actual object assigned to stringHistory

```
stringHistory.display();
```

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

Why instance of is provided: The case for

```
StringHistory stringHistory;
```

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

We have fattened the class with display - we want skinny classes in which separable functions should be in separate classes

```
stringHistory.display()
```

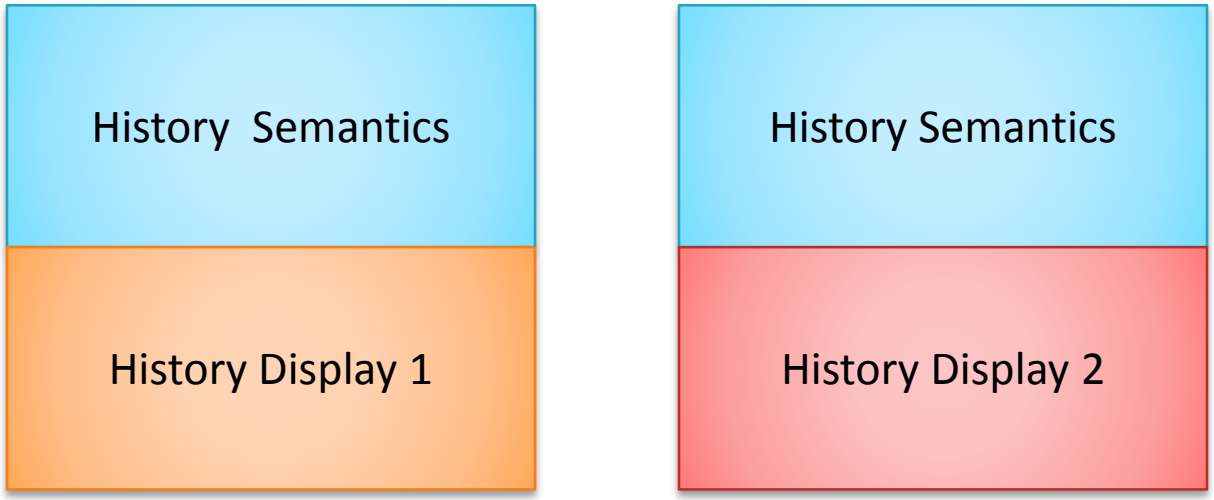
Display and data should be separate, here a separate class handles display of all kinds of string histories

Sometimes separation of concerns requires instanceof

Use instanceof for testing the class of tokens

Scanning and parsing are separate

Separation of Concerns



Can change display without changing other aspects of history

Display and semantics should go in different classes

if a part A of a class can be changed without changing some other part B of the class, then refactor and put A and B in different classes

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

```
ABMISpreadsheet bmiSpreadsheet = …;
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 = …
AStringDatabase database = (AStringDatabase) stringHistory;
stringHistory = (AStringHistory) stringDatabase;
```

Compile Time vs. Runtime Cast Errors: Interfaces

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

Some subtype of ABMISpreadsheet may implement Point

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

```
ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheetAndPoint();
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)

The rest are extra slides