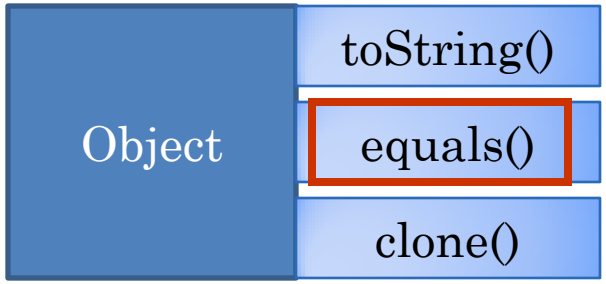
Comp 401
Java Equals and Overloading vs. Polymorphism

Instructor: Prasun Dewan

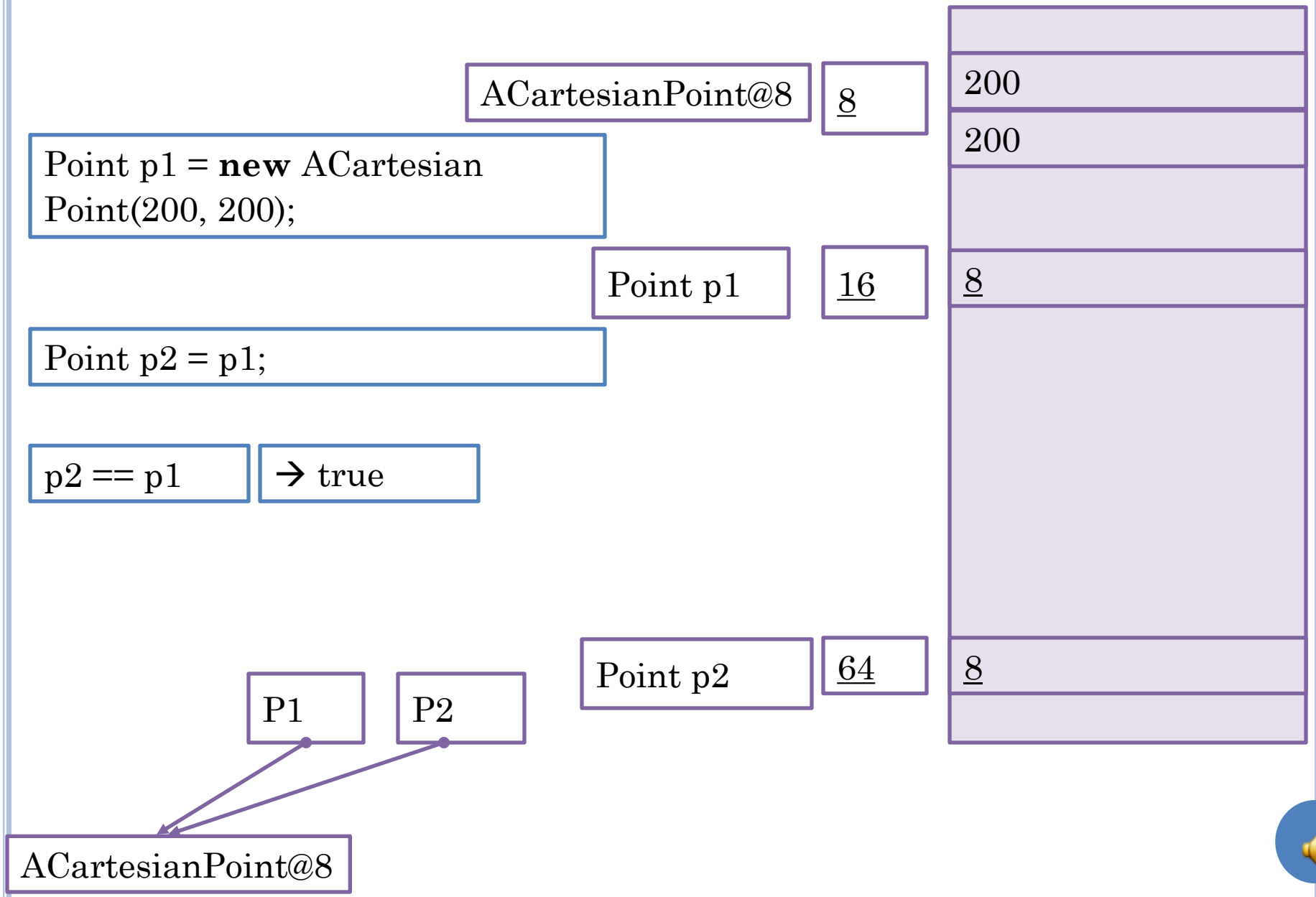
Prerequisite

- Inheritance

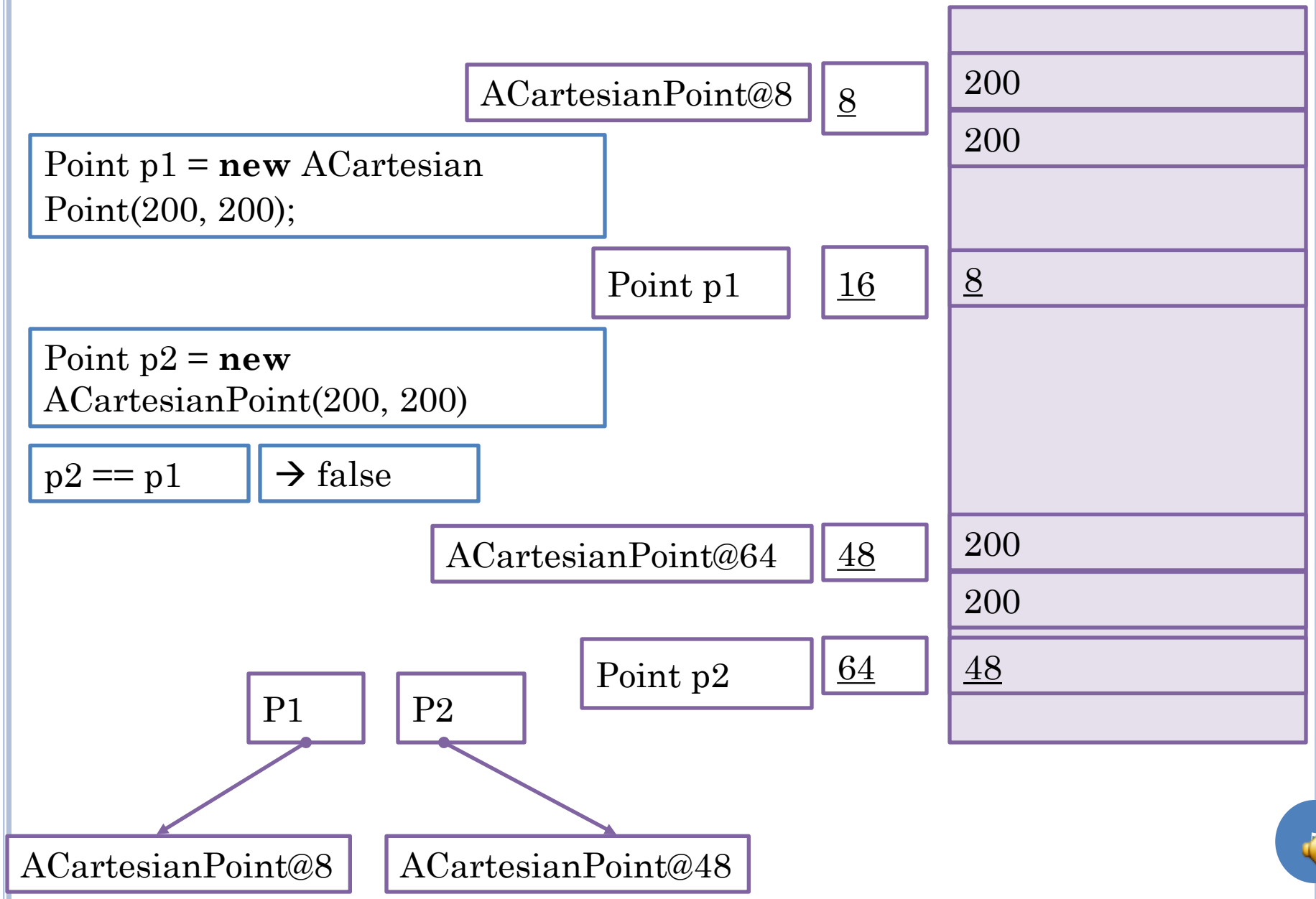
Equals Semantics?



== FOR OBJECTS


ACartesianPoint@8
8
200
200
Point p1 = new ACartesian Point(200, 200);
Point p1
16
8
Point p2 = p1;
p2 == p1
→ true
Point p2
64
8
P1
P2
ACartesianPoint@8

== for Objects


ACartesianPoint@8
8
200
200
Point p1 = new ACartesian Point(200, 200);
Point p1
16
8
Point p2 = new ACartesianPoint(200, 200)
p2 == p1
→ false
ACartesianPoint@64
48
200
200
Point p2
64
48
P1
P2
ACartesianPoint@8
ACartesianPoint@48

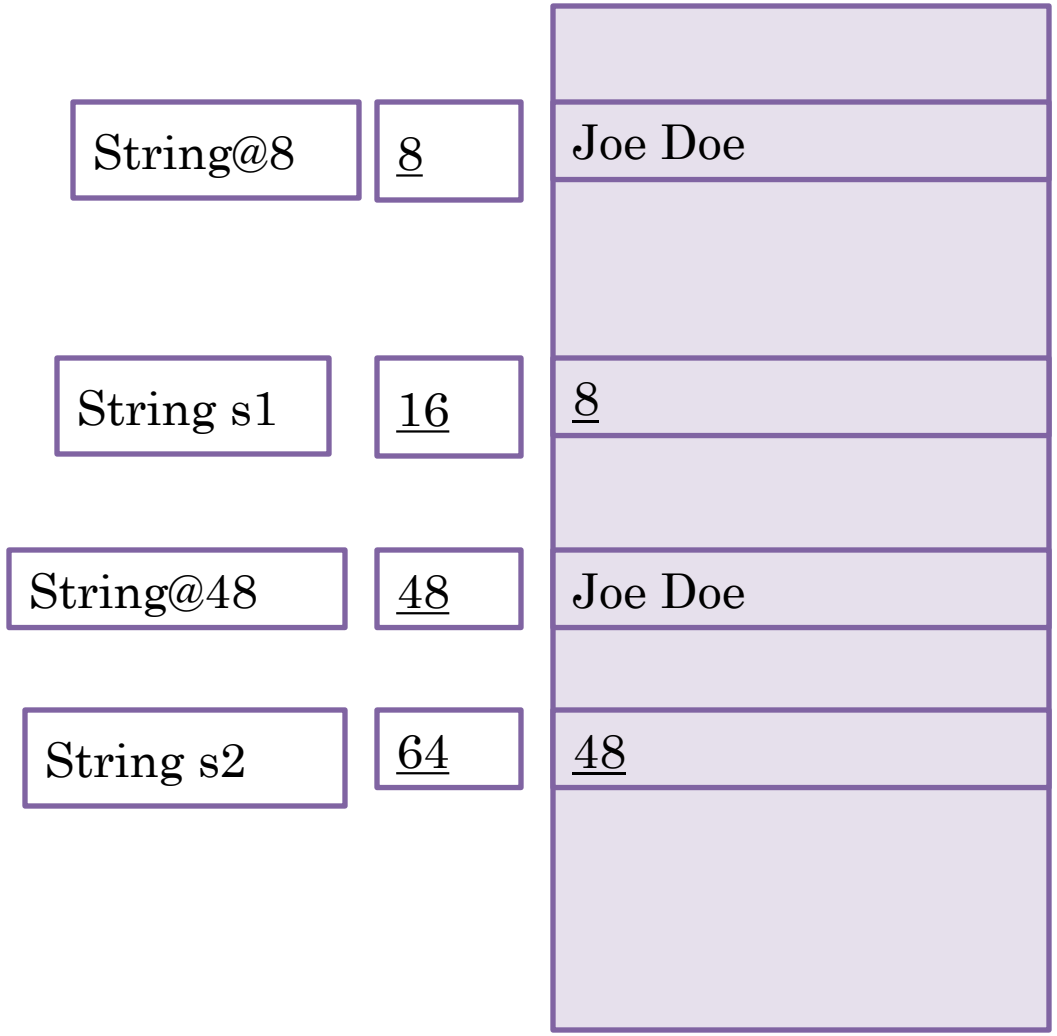
== VS. EQUALS FOR STRINGS

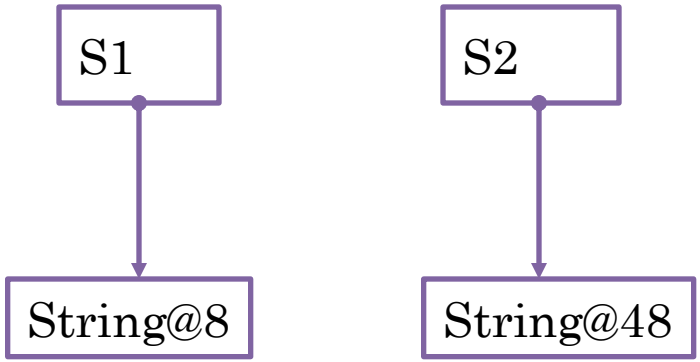
String s1 = **new** String ("Joe Doe");

String s2 = **new** String ("Joe Doe")

s1 == s2 → false

s1.equals(s2) → true





== VS. EQUALS FOR STRINGHISTORY

```
StringHistory stringHistory1 = new AStringHistory();
```

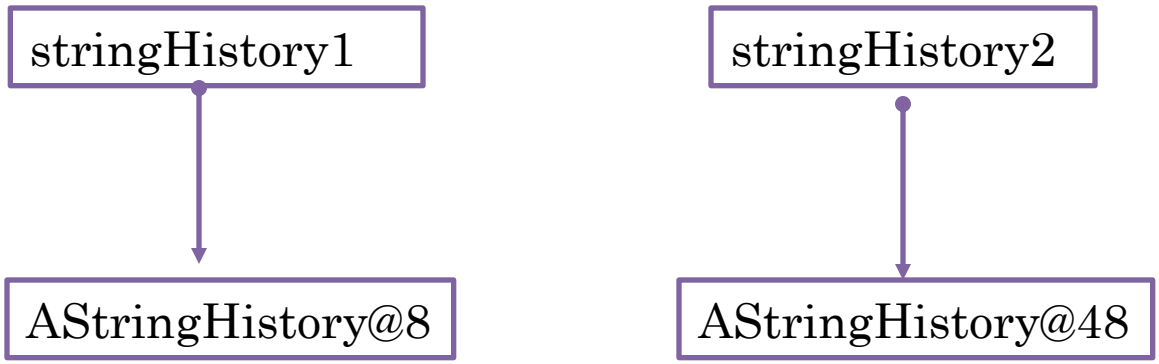
```
StringHistory  stringHistory2  = new AStringHistory();
```

```
stringHistory1 == stringHistory2
```

→ false

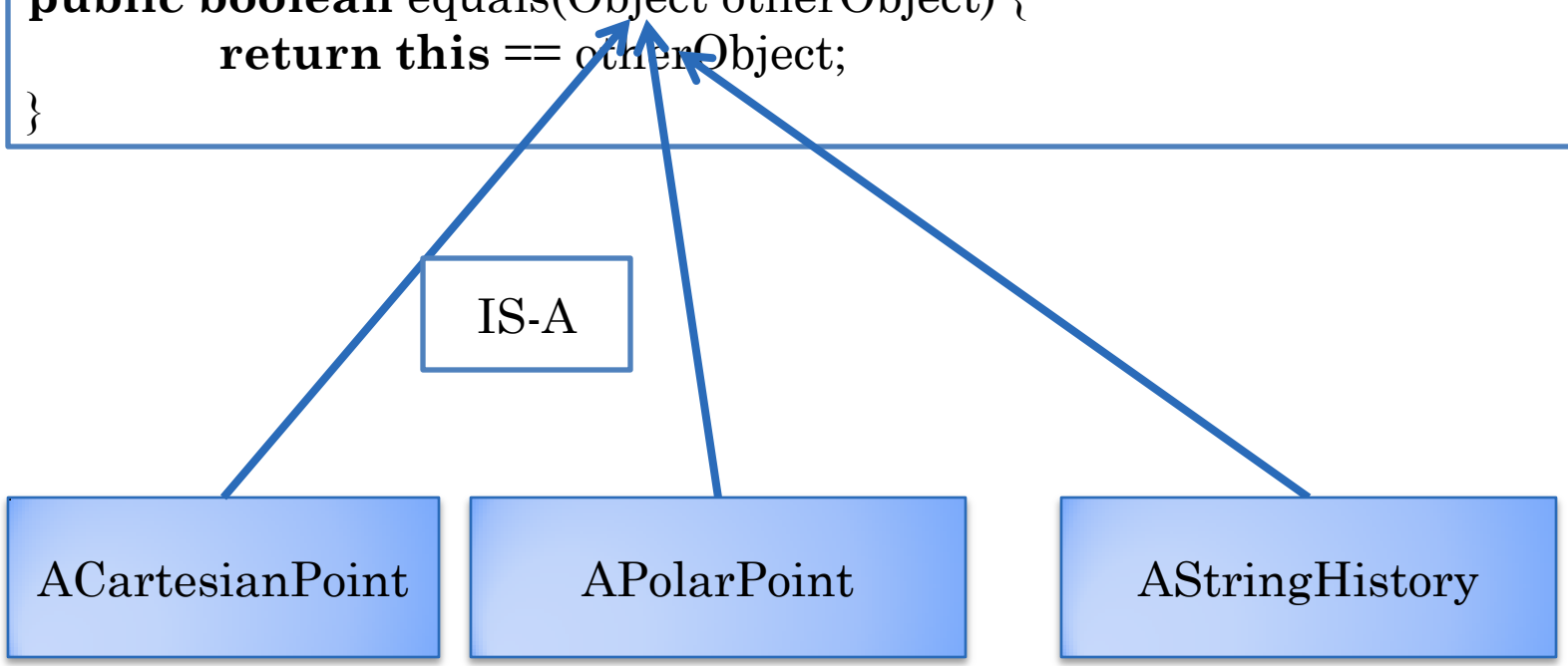
```
stringHistory1.equals(stringHistory2)
```

→ false



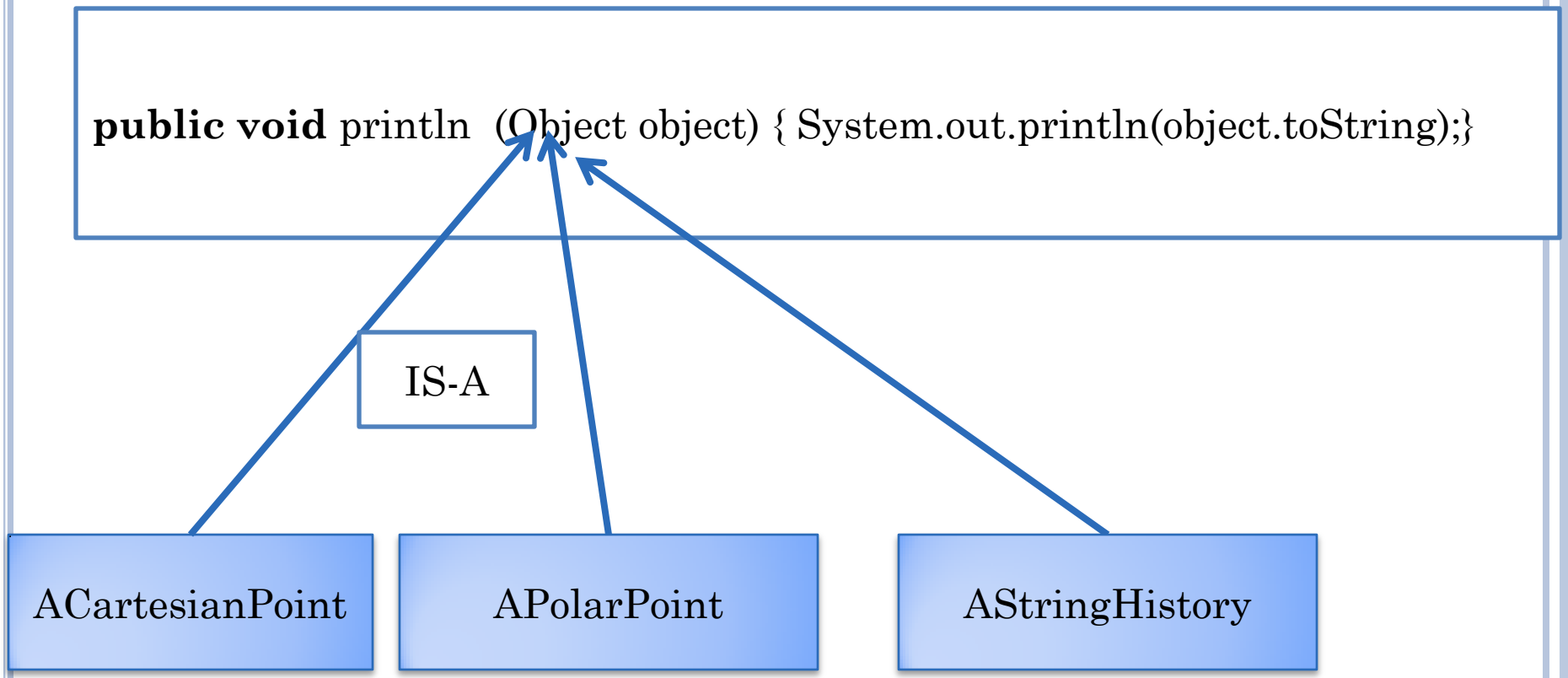
Polymorphic Object Equals Method

```
public boolean equals(Object otherObject) {
        return this == otherObject;
}
```



Polymorphic method: Actual parameters of methods can be of different types

IS-A & Polymorphism



Polymorphic method: Actual parameters of methods can be of different types (instances of different classes)

Overloading vs. Polymorphism

```
public void println  (Object object) {…}
void println  (int i) {…}
void println  (double d) {…}
…
```

Overloaded method: Has same name as some other method in the same class/interface

Polymorphic History Print

```
void print(StringHistory strings) {
    System.out.println("**********");
    for ( int elementNum = 0; elementNum < strings.size(); elementNum++)
        System.out.println(strings.elementAt(elementNum));
}
```

AStringHistory IS-A StringHistory

AStringDatabase IS-A AStringHistory

AStringDatabase IS-A StringHistory

StringSet Users Need Proper equals

```
public boolean equals(Object otherObject) {
        return this == otherObject;
}
```

Override in which class?

Signature of Overridden method ?

Overriding vs. Overloading Equals Method

```
public boolean equals(Object otherObject) {
        …
}
```

```
public boolean equals(StringHistory otherObject) {
        …
}
```

Should be defined in AStringHistory as it is then available to and overriddable by all subclasses.

Signature of Overridden method should be identical in overriding class

Overriding Equals Method (Edit in Class)

```
public boolean equals(Object otherObject) {

}
```

Overriding Equals Method (Edited in Class)

```
public boolean equals(Object otherObject) {

}
```

```
public interface  StringHistory  {
        public void addElement(String element);
        public int size();
        public String elementAt(int index);
}
```

Overriding Equals Method

```
public boolean equals(Object otherObject) {
      if (otherObject == null || !(otherObject instanceof  AStringHistory))
            return false;
       AStringHistory  otherStringHistory = (AStringHistory) otherObject;
       if (size != otherStringHistory.size)
            return false;
       for (int index  = 0; index < size; index++)
            if (!contents[index].equals(otherStringHistory.contents[index]))
                        return false;
       return true;
}
```

O instanceof T == class of O IS-A T

Can be used to compare subclasses also (e.g AStringDatabase and AStringSet)

Variables of this and other instances of a class visible in class

Class-Based Equals Method

```
public boolean equals(Object otherObject) {
      if (otherObject == null  || !(otherObject instanceof  AStringHistory))
            return false;
      AStringHistory  otherStringHistory = (AStringHistory) otherObject;
      if (size != otherStringHistory.size)
            return false;
      for (int index  = 0; index < size; index++)
            if (!contents[index].equals(otherStringHistory.contents[index]))
                  return false;
      return true;
}
```

Cannot be used to compare arbitrary implementation of StringHistory.

StringSet Users Need Proper equals (Review)

```
public boolean equals(Object otherObject) {
        return this == otherObject;
}
```

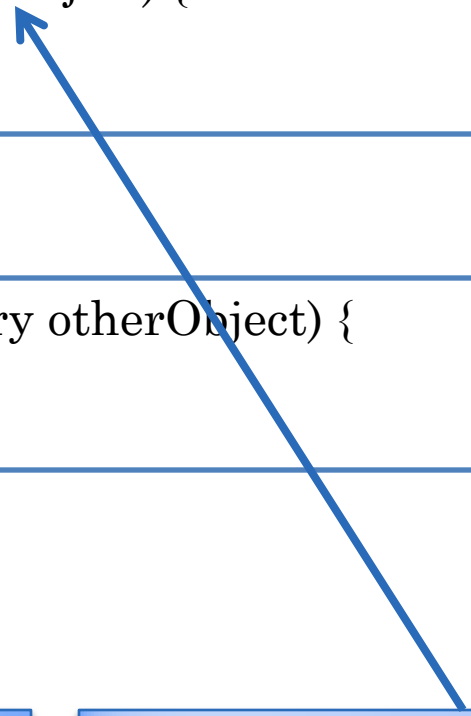
Override in which class?

Signature of Overridden method ?

Overriding vs. Overloading Equals Method (Review)

```
public boolean equals(Object otherObject) {
        …
}
```

```
public boolean equals(StringHistory otherObject) {
        …
}
```



Should be defined in AStringHistory as it is then available to and overriddable by all subclasses.

Signature of Overridden method should be identical in overriding class

Overriding Equals Method (Edited in Class) (Review)

```
public boolean equals(Object otherObject) {
        if (!otherObject instanceof  StringHistory)) return false;
        if (!((StringHistory) otherObject).size() == size()) return false;
         for (int I = 0; I <size(); i++) {
              if (elementAt(i) != (….).elementAt(i) ) return false;
         return true;

}
```

```
public interface  StringHistory  {
            public void addElement(String element);
            public int size();
            public String elementAt(int index);
}
```

Overriding Equals Method (Review)

```
public boolean equals(Object otherObject) {
      if (otherObject == null || !(otherObject instanceof  AStringHistory))
            return false;
       AStringHistory  otherStringHistory = (AStringHistory) otherObject;
       if (size != otherStringHistory.size)
            return false;
       for (int index  = 0; index < size; index++)
            if (!contents[index].equals(otherStringHistory.contents[index]))
                        return false;
       return true;
}
```

O instanceof T == class of O IS-A T

Can be used to compare subclasses also (e.g AStringDatabase and AStringSet)

Variables of this and other instances of a class visible in class

Class-Based Equals Method (Review)

```
public boolean equals(Object otherObject) {
    if (otherObject == null  || !(otherObject instanceof  AStringHistory))
        return false;
    AStringHistory otherStringHistory = (AStringHistory) otherObject;
    if (size != otherStringHistory.size)
        return false;
    for (int index  = 0; index < size; index++)
        if (!contents[index].equals(otherStringHistory.contents[index]))
            return false;
    return true;
}
```

Cannot be used to compare arbitrary implementation of StringHistory.

Interface-Based Equals Method

```
public boolean equals(Object otherObject) {
      if (otherObject == null  ||  !(otherObject instanceof  StringHistory))
            return false;
      StringHistory otherStringHistory = (StringHistory) otherObject;
      if (size != otherStringHistory.size())
            return false;
      for (int index  = 0; index < size; index++)
            if (!contents[index].equals(otherStringHistory.elementAt(index)))
                  return false;
      return true;
}
```

Access variables only if necessary (because of access rules) or efficient

Always type using interfaces!

Can be used to compare arbitrary string histories (e.g AStringHistory and AStringSet)

Accessing methods of otherObject

Less Efficient Interface-Based Equals Method

```
public boolean equals(Object otherObject) {
      if (otherObject == null  || !(otherObject instanceof  StringHistory))
            return false;
       StringHistory  otherStringHistory = (StringHistory) otherObject;
       if (size() != otherStringHistory.size())
            return false;
       for (int index  = 0; index < size(); index++)
            if (!contents.elementAt(index).equals(otherStringHistory.elementAt(index)))
                        return false;
       return true;
}
```

Very inefficient if size counts non-null entries!

Overloading Instead of Overriding

```
public boolean equals(StringHistory  otherStringHistory) {
        if (otherStringHistory == null   | |  size != otherStringHistory.size())
                return false;
        for (int index  = 0; index < size; index++)
                if (!contents[index].equals(otherStringHistory.elementAt(index)))
                        return false;
        return true;
}
```

An overloaded and not overriding method as signature is different from one in Object.

No need to use instanceof

Overriding Equals

```
// in Object
public boolean equals (Object object) {
        …
}
```

```
// in AStringHistory
public boolean equals (Object stringHistory) {
        …
}
```

```
stringHistory.equals(new AStringSet());
```

Equals in most specific class chosen.

Inheritance and Overload resolution

```
// in AStringHistory
public boolean equals (Object object) {
        …
}
```

```
// in AStringHistory
public boolean equals (StringHistory stringHistory) {
        …
}
```

```
stringHistory.equals(new AStringSet());
```

Equals with most specific parameter types chosen

Inheritance and Overload resolution

```
// in AStringSet
public boolean equals (Object object) {
        …
}
```

```
// in AStringHistory
public boolean equals (StringHistory stringHistory) {
        …
}
```

```
stringHistory.equals(new AStringSet());
```

Equals with most specific parameter types chosen, even if it is in less specific class

Overriding vs. Overloading Equals Method in AStringHistory

```
public boolean equals(Object otherObject) {
        …
}
```

```
public boolean equals(StringHistory otherObject) {
        …
}
```

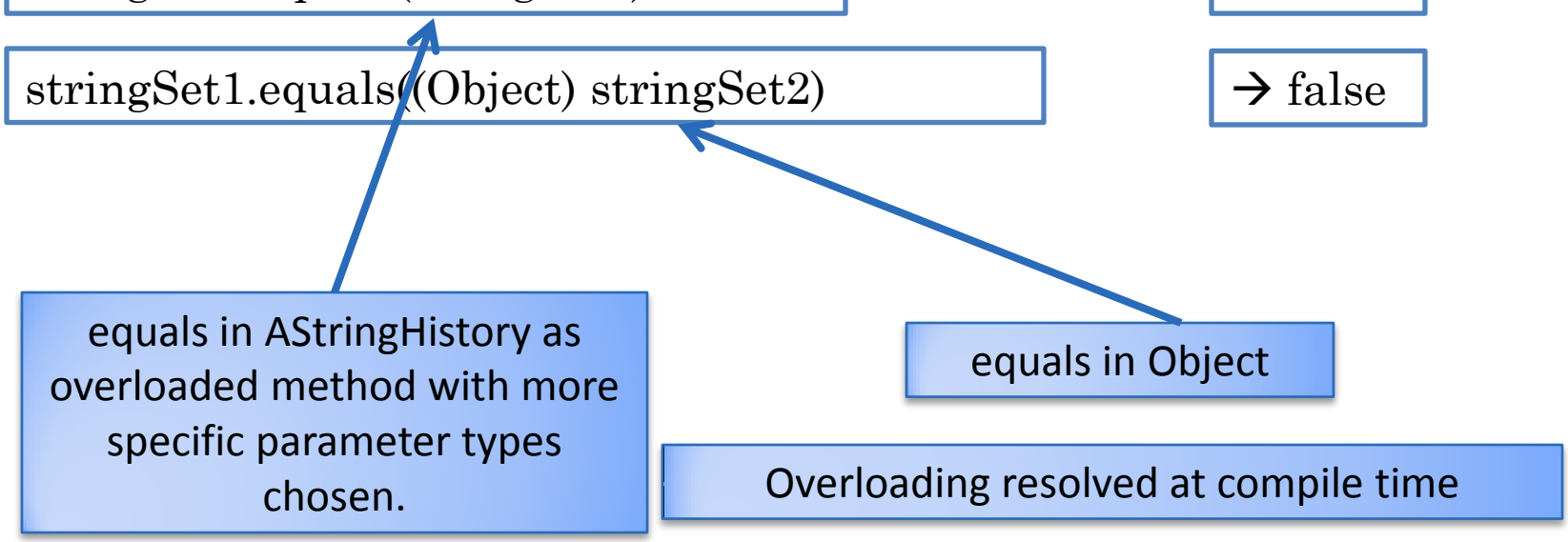
Can the behavior be different if we overload Object equals() instead of override?

Different Behavior

```
StringSet stringSet1 = new AStringSet();
StringSet stringSet2 = new AStringSet();
```

stringSet1.equals(stringSet2) → true

stringSet1.equals((Object) stringSet2) → false



Inheritance and Overload resolution

```
public boolean equals (StringHistory stringHistory1,
                       Object stringHistory2) {
        …
}
```

```
public boolean equals (Object stringHistory1,
                       StringHistory stringHistory2) {
        …
}
```

```
equals(new AStringSet(), new AStringSet());
```

Ambiguous even though Java allows both methods to be legally defined as in other calls there may not be ambiguity!

Inheritance and Overload resolution

```
public boolean equals (StringHistory stringHistory1,
                       Object stringHistory2) {
        …
}
```

```
public boolean equals (Object stringHistory1,
                       StringHistory stringHistory2) {
        …
}
```

```
public boolean equals (StringHistory stringHistory1,
                       StringHistory stringHistory2) {
        …
}
```

```
equals(new AStringSet(), new AStringSet());
```

Equals with most specific parameter types chosen.

Null is an instance of all Types

```
public boolean equals (StringHistory stringHistory1,
                       Object stringHistory2) {
        …
}
```

```
public boolean equals (Object stringHistory1,
                       StringHistory stringHistory2) {
        …
}
```

```
public boolean equals (StringHistory stringHistory1,
                       StringHistory stringHistory2) {
        …
}
```

```
equals(null, null);
```

Equals with most specific parameter types chosen.

Null is an instance of all Types

```
public boolean equals (Point point1, Point point2) {
        …
}
```

```
public boolean equals (StringHistory stringHistory1,
                       StringHistory stringHistory2) {
        …
}
```

```
equals(null, null);
```

Ambiguous!

Typing Null

```
public boolean equals (Point point1, Point point2) {
        …
}
```

```
public boolean equals (StringHistory stringHistory1,
                       StringHistory stringHistory2) {
        …
}
```

```
equals ((StringHistory) null, (StringHistory) null);
```

Null parameters often passed for optional values.

Overriding vs. Overloading Equals Method in AStringHistory

```
public boolean equals(Object otherObject) {
        …
}
```

```
public boolean equals(StringHistory otherObject) {
        …
}
```

The behavior can be different if we overload Object equals() instead of override.

Preventing Overriding Mistakes

```
public boolean equals(StringHistory  otherStringHistory) {
        if (size != otherStringHistory.size())
            return false;
       for (int index  = 0; index < size; index++)
            if (!contents[index].equals(otherStringHistory.elementAt(index))
                    return false;
       return true;
}
```

Suppose we really want to override

Can accidentally create overloaded rather than overriding method.

Preventing Overriding Mistakes

```
@Override
public boolean equals(StringHistory  otherStringHistory) {
        if (size != otherStringHistory.size())
            return false;
        for (int index  = 0; index < size; index++)
            if (!contents[index].equals(otherStringHistory.elementAt(index))
                    return false;
        return true;
}
```

@ indicates an annotation

Annotation is a typed "comment" that can be processed by some tool such as compiler, Eclipse or ObjectEditor at compile/execution time

Compiler will give an error when override assertion not true

Overriding Interface Methods?

```
public class AStringHistory implements StringHistory {
        public final int MAX_SIZE = 50;
        String[] contents = new String[MAX_SIZE];
        int size = 0;
        public int size() { return size;}
        @Override
        public String elementAt (int index) {  return contents[index]; }
        boolean isFull() { return size == MAX_SIZE; }
        public void addElement(String element) {
            if (isFull())
                    System.out.println("Adding item to a full history");
            else {
                    contents[size] = element;
                    size++;
            }
        }
}
```

An overridden method need nor be annotated

Override annotation for method M in class C indicates that M is a(n) (re)implementation of some M declared in some type T, where C IS-A T

Better But Not Supported Annotation

```
public class AStringHistory implements StringHistory {
        public final int MAX_SIZE = 50;
        String[] contents = new String[MAX_SIZE];
        int size = 0;
        public int size() { return size;}
        @Implements
        public String elementAt (int index) {  return contents[index]; }
        boolean isFull() { return size == MAX_SIZE; }
        public void addElement(String element) {
           if (isFull())
                   System.out.println("Adding item to a full history");
           else {
                   contents[size] = element;
                   size++;
           }
    }
}
```

Overriding vs. Overloading Equals Method in AStringHistory

```
public boolean equals(Object otherObject) {
        …
}
```

```
public boolean equals(StringHistory otherObject) {
        …
}
```

The behavior can be different if we overload Object equals() instead of override.

Can accidentally create overloaded rather than overriding method.

Overloading a mistake?

Overloading a Mistake?

```
public boolean equals(Object otherObject) {
      if (!(otherObject instanceof  StringHistory))
            return false;
       StringHistory  otherStringHistory = (StringHistory) otherObject;
       if (size != otherStringHistory.size())
            return false;
       for (int index  = 0; index < size; index++)
            if (!contents[index].equals(otherStringHistory.elementAt(index))
                  return false;
       return true;
}
```

Use instanceof only if necessary as it creates messy and difficult to extend code (more on this later)

```
stringHistory1.equals((Object) stringHistory2)
```

Programmer probably wants Object equals()