COMP 401
INHERITANCE: TYPE CHECKING

Instructor: Prasun Dewan
Prerequisite

- Inheritance
TYPE-CHECKING EXAMPLES

StringHistory stringHistory = new AStringDatabase();

StringDatabase stringDatabase = new AStringHistory();
**Type-Checking Examples**

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

stringDatabase.clear();
```

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

stringHistory.size();

stringHistory.clear();
```

```java
((StringDatabase) stringHistory).clear();
```
## Type-Checking Examples

<table>
<thead>
<tr>
<th>Code</th>
<th>Type Mismatch</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>StringDatabase stringDatabase = new AStringHistory();</code></td>
<td></td>
</tr>
<tr>
<td><code>stringDatabase.clear()</code></td>
<td></td>
</tr>
<tr>
<td><code>StringHistory stringHistory = new AStringDatabase();</code></td>
<td></td>
</tr>
<tr>
<td><code>stringHistory.size()</code></td>
<td></td>
</tr>
<tr>
<td><code>stringHistory.clear()</code></td>
<td></td>
</tr>
<tr>
<td><code>((StringDatabase) stringHistory).clear()</code></td>
<td></td>
</tr>
</tbody>
</table>

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

myCar.steer();

myCar.setCity("Raleigh");

Regular Model Assigned
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

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

```java
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) bmiSpreadsheet

**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) bmiSpreadsheet;

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 ABMISpreadsheetAndPoint();
Point cartesianPoint = (Point) bmiSpreadsheet;
bmiSpreadsheet = (ABMISpreadsheet) cartesianPoint;
**Final Class**

String string = “hello”;
Point cartesianPoint = (Point) string;

Some subtype of String may implement Point?

```java
public class AStringAndPoint
    extends String
    implements Point {
...
}
```

String is a final class and thus cannot be subtyped

```java
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: \( T_1 \) extends \( T_2 \) \( \Rightarrow \) \( T_1 \) IS-A \( T_2 \)
- Implements: \( T_1 \) implements \( T_2 \) \( \Rightarrow \) \( T_1 \) IS-A \( T_2 \)
- Transitive:
  - \( T_1 \) IS-A \( T_2 \)
  - \( T_2 \) IS-A \( T_3 \)
  - \( \Rightarrow \) \( T_1 \) IS-A \( T_3 \)
- Reflexive:
  - \( T_1 == T_2 \) \( \Rightarrow \) \( T_1 \) IS-A \( T_2 \)

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

Special rule saying very type IS-A Object
CONCEPTUAL PROBLEMS IN JAVA

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

```java
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 use instanceof

Usually considered bad programming to decide alternatives based on 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 instanceof**

```java
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`:

```java
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`**

1. Use `instanceof` for testing the class of tokens.
2. Scanning and parsing are separate.
3. Display and data should be separate.
4. Sometimes separation of concerns requires `instanceof`.

```java
StringHistory stringHistory;

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

stringHistory.display();
```
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

How does program distinguish between different input characters?
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());
    }
}
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);
    }
}
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);
            }
        }
    }
}
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);
            }
    }
}
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);
        }
    }
}
**Illegal Switch**

Type of switch expression must be ordinal type

```java
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**

StringHistory stringHistory1, stringHistory2;

stringHistory1.equals(stringHistory2);

Object operation special cased.

All interfaces should inherit from interface defined by Object..
Garbage Collection

Point p1 = new ACartesianPoint(100,100);

Point p2 = new ACartesianPoint(150,75);

p2 = p1;

Garbage Collected
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
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
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 newVal);
    public int getY();
    public void setY(int newVal);
    public Point getLocation();
}

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

```java
@StructurePattern(StructurePatternNames.POINT_PATTERN)
public interface Point {
    public int getX();
    public int getY();
    public double getAngle();
    public double getRadius();
}
```
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)
public class ABoundedPoint extends ACartesianPoint implements Point {
    Point upperLeftCorner, lowerRightCorner;
    public ABoundedPoint(int initX, int initY, Point initUpperLeftCorner, Point initLowerRightCorner) {
        super(initX, initY);
        x = Math.max(x, upperLeftCorner.getX());
        x = Math.min(x, lowerRightCorner.getX());
        y = Math.max(y, upperLeftCorner.getY());
        y = Math.min(y, lowerRightCorner.getY());
    }
}

Calling Constructor in Superclass

Superclass constructor

Variables also extended unlike in collection example

Slide to be deleted
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

```java
public class AStringHistoryAndPoint
    extends AStringHistory implements Point {
    ...
}
```

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

```java
String s = new AStringAndPoint();
Point point = (Point) string;
```

```java
public class AStringAndPoint
    extends String implements Point {
    ...
}
```

```java
public final class String {
    ...
}
```