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
  - A CartesianPoint IS-A Point

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

Object

String History

String Database

AString History

AString Database

implements

extends
IS-A Definition

- **Implements:** $T_1$ implements $T_2$ $\Rightarrow$ $T_1$ IS-A $T_2$
  - `AStringHistory` IS-A `StringHistory`
  - `AStringDatabase` IS-A `StringDatabase`

- **Extends:** $T_1$ extends $T_2$ $\Rightarrow$ $T_1$ IS-A $T_2$
  - `StringDatabase` IS-A `StringHistory`
  - `AStringDatabase` IS-A `AStringHistory`

- **Transitive:**
  - $T_1$ IS-A $T_2$
  - $T_2$ IS-A $T_3$
  - $\Rightarrow$ $T_1$ IS-A $T_3$
    - `AStringDatabase` IS-A `StringHistory`

- **Reflexive:**
  - $T_1$ == $T_2$ $\Rightarrow$ $T_1$ IS-A $T_2$
    - `StringHistory` IS-A `StringHistory`
**Why Interesting?**

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

Object
- AString History
  - AString Database
- String History
  - String Database

Helps in understanding the designs/implementations

Helps users of the types
**Exact Typing and Overloaded Methods**

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

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

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

```java
print (new AnotherBMISpreadsheet());
```
FLEXIBLE TYPING AND POLYMORPHISM

```java
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
  - A CartesianPoint IS-A Point

\[ T^1 \text{ IS-A } T^2 \text{ if } T^1 \text{ has all traits (methods and variables in Java) of } T^2 \]

\[ T^1 \text{ IS-A } T^2 \text{ if the set of members of } T^1 \text{ is a subset of the set of members of } T^2 \]

\[ T^1 \text{ IS-A } T^2 \text{ if } T^2 \text{ can be substituted by } T^1 \text{ that is whenever a member of } T^2 \text{ is expected a member of } T^1 \text{ can be used} \]
IS-A RELATIONSHIP (REVIEW)

- String History implements String Database
- AString History extends String Database
- Object
  - String
  - AString History
  - String Database

Relationships:
- Implements: String History
- Extends: AString History
Why Interesting? (Review)

Physical Object
- Animal
  - Mammal
    - Primate
      - Human
- Vehicle
- Object
- String
- Database

Helps in understanding the designs/implementations

$T^1 \text{ IS-A } T^2$ if $T^1$ has all traits (methods and variables in Java) of $T^2$

Helps users of the types

$T^1 \text{ IS-A } T^2$ 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
Flexible Typing and Polymorphism (Review)

```java
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 T2 can be substituted by T1 that is whenever a member of T2 is expected a member of T1 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()}
If you need to be assigned a vehicle (a pet) a car or bicycle (a cat or dog) will do

If you need to be assigned a car (a dog) a bicycle (a cat) will not do
**GETTING AN UPGRADE**

Regular Model Requested

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

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");
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

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

println takes argument of type Object

! (Interface subtype of Class)

Interface subtype of Object
TYPE-CHECKING EXAMPLES FOR PRIMITIVE TYPES

int i = 2.5;  

double d = 2;  

int i = (int) 2.5;
**Assignment Rules for Primitive Types**

- If $T_1$ narrower than $T_2$ (Set of instances of $T_1 \subseteq$ Set of instances of $T_2$)
- Expression of type $T_1$ can be assigned to variable of type $T_2$
- Expression of type $T_2$ can be assigned to variable of type $T_1$ with cast
  - Primitive casts can convert values
    - `(int)` double converts double to int value
  - Object casts never convert values!
DAG? MULTIPLE INHERITANCE?

- Human
- Vehicle
- Parent
- AString History
- ALocatable
- ALocatableStringHistory
**Problem with Multiple Class Inheritance**

Which `toString()` should be called? They are different methods with the same header.
**Problem with Multiple Interface Inheritance?**

- **StringHistory**
  - `public String toString();`

- **Locatable**
  - `public String toString();`

- **LocatableStringHistory**

Duplicates added to a set of methods

Inheriting the fact you have eyes vs. a particular kind of eyes
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.
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);
    }
}
public static void main (String[] args) {
    BMISpreadsheet bmiSpreadsheet = new ABMISpreadsheetAndCalculator();
    bmi = bmiSpreadsheet.getBMI();
    // bmi = bmiSpreadsheet.calculateBMI(1.77, 75);
}
An interface can extend multiple interfaces

A class cannot extend multiple classes (in Java)

An extended interface or class may not have extra methods!
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);
    }
}
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();
}
INHERITING CONSTANTS

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

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

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

Ambiguous.
INHERITING CONSTANTS

```java
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**

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

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

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

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

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

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

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

public class AMutablePoint extends ACartesianPoint implements Point {...}

Ambiguous

@StructurePattern(StructurePatternNames.RECTANGLE_PATTERN)
public class AMutablePoint extends ACartesianPoint implements MutablePoint {...}
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.
If type $T_1$ contains a super set of the method headers in another type $T_2$, then $T_1$ should extend $T_2$?
**Correct Use of Inheritance**

- **String History**
  - `size()`
  - `elementAt()`
  - `addElement()`

- **AString History**
  - `size`
  - `contents`
  - `MAX_SIZE`
  - `isFull()`

- **String Database**
  - `member()`
  - `removeElement()`
  - `clear()`

- **AString Database**
  - `removeElement(int)`
  - `shiftUp()`
  - `indexOf()`

- **Supertype**
  - `extends`

- **Subtype**
  - `implements`

- **Superclass**
  - `extends`

- **Subclass**
  - `extends`
NOT USING INHERITANCE WHEN YOU CAN

Method header duplication

size()

String History

size

contents

AString History

size()

elementAt()

addElement()
Not Using Inheritance When you can?

String History
- size()
- elementAt()
- addElement()

Same conceptual operation associated with different header

AString History
- size
- contents
- MAX_SIZE
- isFull()

Same code associated with different header

String Database
- sizeD()
- elementAtD()
- addElementD()
- memberD()
- removeElementD()
- clearD()

AString Database
- size
- contents
- MAX_SIZE
- removeElement(int)
- shiftUp()
- indexOf()
- isFullD()
**Point and Line**

- **Point**
  - `getX()`
  - `getY()`
  - `setX()`
  - `setY()`

- **Line**
  - `getX()`
  - `getY()`
  - `setWidth()`
  - `setHeight()`
  - `setX()`
  - `setY()`
  - `setWidth()`
  - `setHeight()`
INHERIT?

A Line is not a Point!

T1 IS-A T2 if T2 can be substituted by T1 that is whenever a member of T2 is expected a member of T1 can be used
INHERIT? (REVIEW)

A Line is not a Point!

T¹ IS-A T² if T² can be substituted by T¹ that is whenever a member of T² is expected a member of T¹ can be used.
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.
A Vertical Line

T¹ IS-A T² if the set of members of T¹ is a subset of the set of members of T²
**Overriding Code and Limitations**

### Public Class `AVerticalLine` extends `ALine`

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

### Public static boolean `checkLine(Line aLine, int aWidth)`

```java
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`

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

$T^1 \text{ IS-A } T^2$ 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?

A Line is not a vertical or horizontal line!

From bad to worse
Refactor?

No principle violations

Perhaps awkward and too complicated
Give an Error?

Check will fail but checker knows the failure is deliberate

We have made a “bug” a “feature”

```java
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
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

```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 instanceof makes more sense
### Alternatives

StringHistory stringHistory;

```java
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**

```java
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`

```java
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

<table>
<thead>
<tr>
<th>StringHistory stringHistory;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>if</strong> (stringHistory <em>instanceof</em> StringDatabase) {</td>
</tr>
<tr>
<td>display((StringDatabase) stringHistory);</td>
</tr>
<tr>
<td>} <strong>else if</strong> (stringHistory <em>instanceof</em> StringSet) {</td>
</tr>
<tr>
<td>display((StringSet) stringHistory);</td>
</tr>
<tr>
<td>} <strong>else</strong> {</td>
</tr>
<tr>
<td>display(stringHistory);</td>
</tr>
</tbody>
</table>
| }

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

- 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

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

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

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) 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 = ... 
AStringDatabase database = (AStringDatabase) stringHistory;
stringHistory = (AStringHistory) stringDatabase;
Compile Time vs. Runtime Cast Errors: Interfaces

Some subtype of ABMISpreadsheet may implement Point

```java
public class ABMISpreadsheetAndPoint
    extends ABMISpreadsheet
    implements Point {
    ...
}
```

```java
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)
The rest are extra slides