COMP 401: INTERFACES

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
public class Factorials {
    public static long loopingFactorial(int n) {
        long product = 1;
        while (n > 0) {
            product *= n;
            n -= 1;
        }
        return product;
    }
    ...
}
public class Factorials {
    public static long recursingFactorial(int n) {
        if (n <= 1)
            return 1;
        return n * recursingFactorial(n - 1);
    }
}
public class ALoopingFactorialSpreadsheet {
    int number;
    long factorial;
    public int getNumber() {
        return number;
    }
    public void setNumber(int newVal) {
        number = newVal;
        factorial = Factorials.loopingFactorial(number);
    }
    public long getFactorial() {
        return factorial;
    }
}

ALoopingFactorialSpreadsheet factorial1 =
    new ALoopingFactorialSpreadsheet ();
ALoopingFactorialSpreadsheet factorial2 =
    new ALoopingFactorialSpreadsheet ();
factorial1.setNumber(2);
factorial2.setNumber(7);
public class ARecursingFactorialSpreadsheet {
    int number;
    long factorial;
    public int getNumber() {
        return number;
    }
    public void setNumber(int newVal) {
        number = newVal;
        factorial = Factorials.recursingFactorial(number);
    }
    public long getFactorial() {
        return factorial;
    }
}

ARecursingFactorialSpreadsheet factorial1 = new ARecursingFactorialSpreadsheet();
ARecursingFactorialSpreadsheet factorial2 = new ARecursingFactorialSpreadsheet();
factorial1.setNumber(2);
factorial2.setNumber(7);

Uses looping rather than recursive factorial

User does not know or care

The public methods (visible components) of the two classes remain the same

Method calling code remains the same
Interfaces: Capturing Common Visible Components

```java
public interface FactorialSpreadsheet {
    public int getNumber();
    public void setNumber(int newVal);
    public long getFactorial();
}
```

Like a class, an interface must be in its own file, with a .java suffix.
public class ALoopingFactorialSpreadsheet implements FactorialSpreadsheet {
    int number;
    long factorial;
    public int getNumber() {
        return number;
    }
    public void setNumber(int newVal) {
        number = newVal;
        factorial = Factorials.loopingFactorial(number);
    }
    public long getFactorial() {
        return factorial;
    }
}

FactorialSpreadsheet factorial1 = new ALoopingFactorialSpreadsheet();
FactorialSpreadsheet factorial2 = new ALoopingFactorialSpreadsheet();
factorial1.setNumber(2);
factorial2.setNumber(2);
**Another Implementation of Same Interface**

```java
public class ARecursingFactorialSpreadsheet implements FactorialSpreadsheet {
    int number;
    long factorial;
    public int getNumber() {
        return number;
    }
    public void setNumber(int newVal) {
        number = newVal;
        factorial = Factorials.recursingFactorial(number);
    }
    public long getFactorial() {
        return factorial;
    }
}
```

A variable typed by an interface can be assigned an instance of any class that implements the interface. This makes it easier to switch implementations.
public class ABMISpreadsheet implements BMISpreadsheet {
    double height;
    double weight;

    public ABMISpreadsheet(double theInitialHeight, double theInitialWeight) {
        setHeight(theInitialHeight);
        setWeight(theInitialWeight);
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double newWeight) {
        weight = newWeight;
    }

    public double getHeight() {
        return height;
    }

    public void setHeight(double newHeight) {
        height = newHeight;
    }

    public double getBMI() {
        return weight/(height*height);
    }
}
REAL-WORLD ANALOGY FOR INTERFACES

Specification

implements

manufactured by

manufactured by

implements
public interface BMISpreadsheet {
    double getWeight();
    void setWeight(double newWeight);
    double getHeight();
    void setHeight(double newHeight);
    double getBMI();
}
public class AnotherBMISpreadsheet implements BMISpreadsheet {
    double height, weight, bmi;
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
        bmi = weight/(height*height);
    }
    ...
public class ABMISpreadsheet implements BMISpreadsheet {
    double height;
    double weight;
    public ABMISpreadsheet(double theInitialHeight, double theInitialWeight) {
        setHeight(theInitialHeight);
        setWeight(theInitialWeight);
    }
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
public class AnotherBMISpreadsheet implements BMISpreadsheet{
    double height, weight, bmi;

    public double getHeight() {
        return height;
    }

    public void setHeight(double newHeight) {
        height = newHeight;
        bmi = weight/(height*height);
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double newWeight) {
        weight = newWeight;
        bmi = weight/(height*height);
    }

    public double getBMI() {
        return bmi;
    }
}

Independent value setters compute and store the new dependent properties returned by the property getters.

Instead of dependent property getters computing the values.
public interface BMISpreadsheet {
    double getWeight();
    void setWeight(double newWeight);
    double getHeight();
    void setHeight(double newHeight);
    double getBMI();
}
public class ABMISpreadsheet implements BMISpreadsheet {
    double height;
    double weight;
    public ABMISpreadsheet(double theInitialHeight, double theInitialWeight) {
        setHeight(theInitialHeight);
        setWeight(theInitialWeight);
    }
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
public class AnotherBMISpreadsheet implements BMISpreadsheet{
    double height, weight, bmi;
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
        bmi = weight/(height*height);
    }
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
        bmi = weight/(height*height);
    }
    public double getBMI() {
        return bmi;
    }
}

Independent value setters compute and store the new dependent properties returned by the property getters

Instead of dependent property getters computing the values
**INTERFACE**

- **BMISpreadsheet**
  - implements
  - ABMISpreadsheet

- **AnotherBMISpreadsheet**
  - instance of
  - AnotherBMISpreadsheet instance

- **AnotherBMISpreadsheet**
  - instance of
  - ABMISpreadsheet instance
Java Instanceof Boolean Operator

```
(new ABMISpreadsheet(1.77, 75)) instanceof ABMISpreadsheet  # true

(new ABMISpreadsheet(1.77, 75)) instanceof BMISpreadsheet   # true

(new ABMISpreadsheet(1.77, 75)) instanceof AnotherBMISpreadsheet  # false
```

- true and false are values of Java type boolean
- Instanceof is Java keyword
- If class or interface of object o is T then o instanceof T returns true
Using Interfaces to Type

```java
public class BMISpreadsheetUser {
    public static void main(String[] args) {
        BMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet(1.77, 75);
        System.out.println(bmiSpreadsheet.getBMI());
        bmiSpreadsheet = new AnotherBMISpreadsheet();
        bmiSpreadsheet.setHeight(1.77);
        bmiSpreadsheet.setWeight(75);
        System.out.println(bmiSpreadsheet.getBMI());
    }
}
```

- Interface as type
- Same variable assigned instances of two different classes
public class BMISpreadsheetUser {
    public static void main(String[] args) {
        BMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet(1.77, 75);
        System.out.println(bmiSpreadsheet.obtainBMI());
        bmiSpreadsheet = new AnotherBMISpreadsheet();
        bmiSpreadsheet.setHeight(1.77);
        bmiSpreadsheet.setWeight(75);
        System.out.println(bmiSpreadsheet.getBMI());
    }
}

Not defined in interface (or class)
**Interface Methods Considered in Type Checking**

```java
public class BMISpreadsheetUser {
    public static void main(String[] args) {
        BMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet(1.77, 75);
        System.out.println(bmiSpreadsheet.getBMI());
        bmiSpreadsheet = new AnotherBMISpreadsheet();
        bmiSpreadsheet.setHeight(1.77);
        System.out.println(bmiSpreadsheet.getBMI());
    }
}
```

Not defined in interface so illegal, even though defined in class
Class-based 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());
}

print (new AnotherBMISpreadsheet());
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());
print (new ABMISpreadsheet());
```
### Interface-Based Typing and Polymorphic Methods

```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());
```

Polymorphic method: A method having at least one parameter that can be assigned objects of different classes.
public class ABMISpreadsheetAndCalculator implements ??? {
    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);
    }
}
**Implementing Multiple Interfaces**

```java
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 interface BMICalculator {
    public double calculateBMI(double height, double weight);
}
public class ABMICalculator implements BMICalculator {
    public double calculateBMI(double height, double weight) {
        return weight/(height*height);
    }
}
## Java Instanceof Boolean Operator

| new ABMISpreadsheetAndCalculator() instanceof ABMISpreadsheetAndCalculator | true |
| new ABMISpreadsheetAndCalculator() instanceof BMISpreadsheet | true |
| new ABMISpreadsheetAndCalculator() instanceof BMICalculator | false |
| new ABMISpreadsheetAndCalculator() instanceof ABMICalculator | false |
| new ABMISpreadsheetAndCalculator() instanceof ABMISpreadsheet | false |

If class or interface of object o is T then o instanceof T returns true
public static void main (String[] args) {
    BMICalculator bmiCalculator = new ABMISpreadsheetAndCalculator();
    BMISpreadsheet bmiSpreadsheet = new ABMISpreadsheetAndCalculator();
    double bmi = bmiCalculator.calculateBMI(1.77, 75);
    bmi = bmiSpreadsheet.getBMI(); //
    bmi = bmiCalculator.getBMI(); //
    bmi = bmiSpreadsheet.calculateBMI(1.77, 75);
}

public static void main (String[] args) {
    BMICalculator[] bmiCalculators = {
        new ABMISpreadsheetAndCalculator(),
        new ABMICalculatorWithInterface()};
}
**CAR ANALOGY**

- A car is characterized by
  - Its make
  - License plate
  - Registration

- Licensing authority groups car by the registration or license plate
CANNOT INstantiate SPECIFICATION

- Cannot order a car from a specification
  - Must order from factory
  - A car defined by Corvette specification ordered from factory implementing the specification

- Cannot instantiate interface
  - Must instantiate class
  - BMISpreadsheet instance created by instantiating class implementing interface
**INTERFACE AS A SYNTACTIC SPECIFICATION**

```java
public class ABMISpreadsheet implements BMISpreadsheet{
    double height;
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
```
INTERFACE AS A SYNTACTIC SPECIFICATION

```java
public class ABMISpreadsheet implements BMISpreadsheet{
    double height;
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getBMI() {
        return 13450;
    }
}
```

Syntactic Contract

Bombay Market Index
public class ABMISpreadsheet implements BMISpreadsheet {
    double height;
    double weight;

    public ABMISpreadsheet(double theInitialHeight, double theInitialWeight) {
        setHeight(theInitialHeight);
        setWeight(theInitialWeight);
    }

    public double getHeight() {
        return height;
    }

    public void setHeight(double newHeight) {
        height = newHeight;
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double newWeight) {
        weight = newWeight;
    }

    public double getBMI() {
        return weight/(height*height);
    }
}
public class AnotherBMISpreadsheet implements BMISpreadsheet{

double height, weight, bmi;

public double getHeight() {
    return height;
}

public void setHeight(double newHeight) {
    height = newHeight;
    bmi = weight/(height*height);
}

public double getWeight() {
    return weight;
}

public void setWeight(double newWeight) {
    weight = newWeight;
    bmi = weight/(height*height);
}

public double getBMI() {
    return bmi;
}

}

Independent value setters compute and store the new dependent properties returned by the property getters

Instead of dependent property getters computing the values
public interface BMISpreadsheet {
    double getWeight();
    void setWeight(double newWeight);
    double getHeight();
    void setHeight(double newHeight);
    double getBMI();
}
TOP DOWN: MATHEMATICAL POINT

\[ X, Y \]

\[ R, \theta \]
Point Interface

```java
public interface Point {
    public int getX();
    public int getY();
    public double getAngle();
    public double getRadius();
}
```

Read-only properties defining immutable object
**CLASS: ACartesianPoint**

```java
public class ACartesianPoint implements Point {
    int x, y;
    public ACartesianPoint(int theX, int theY) {
        x = theX;
        y = theY;
    }
    public ACartesianPoint(double theRadius, double theAngle) {
        x = (int) (theRadius*Math.cos(theAngle));
        y = (int) (theRadius*Math.sin(theAngle));
    }
    public int getX() { return x; }
    public int getY() { return y; }
    public double getAngle() { return Math.atan2(y, x); }
    public double getRadius() { return Math.sqrt(x*x + y*y); }
}
```
public class APolarPoint implements Point {
  double radius, angle;
  public APolarPoint(double theRadius, double theAngle) {
    radius = theRadius;
    angle = theAngle;
  }
  public APolarPoint(int theX, int theY) {
    radius = Math.sqrt(theX*theX + theY*theY);
    angle = Math.atan((double) theY/theX);
  }
  public int getX() { return (int) (radius*Math.cos(angle)); }
  public int getY() { return (int) (radius*Math.sin(angle)); }
  public double getAngle() { return angle; }
  public double getRadius() { return radius; }
}
Representing Geometric Objects

- Geometric example to show multiple useful implementations of an interface
- Most geometric objects have multiple representations
**WHAT IS A REPRESENTATION**

- **Logical representation**
  - Defined by its interface
  - Specifies properties
- **Physical representation**
  - Defined by its instance variables
- **ACartesianPoint** and **APolarPoint** have the same logical representation but different physical representation
LINE LOGICAL REPRESENTATION (PROPERTIES)

X, Y (int)

Bounding Rectangle

Width (int)

Height (int)
public interface Line {
    public int getX();
    public void setX(int newX);
    public int getY();
    public void setY(int newY);
    public int getWidth();
    public void setWidth(int newVal); 
    public int getHeight();
    public void setHeight(int newHeight);
}
public class ALine implements Line {
    int x, y, width, height;
    public ALine (int initX, int initY, int initWidth, int initHeight) {
        x = initX;
        y = initY;
        width = initWidth;
        height = initHeight;
    }
    public int getX() {return x;}
    public void setX(int newX) {x = newX;}
    public int getY() { return y; }
    public void setY(int newY) {y = newY;}
    public int getWidth() {return width;}
    public void setWidth(int newVal) {width = newVal;}
    public int getHeight() {return height;}
    public void setHeight(int newHeight) {height = newHeight;}
}
Interfaces

- Define contracts between our users and implementers
- Optional – they may not be used
- Good style to use them
  - Each instance method must be in some interface (static methods cannot be in interfaces)
  - Each object variable must be type by an interface
- Can be derived bottom up or top down
ASSUMES ONLY ONE IMPLEMENTATION OF INTERFACE WILL BE CREATED
Extra
Line
Java Coordinate System

X and Y coordinates must be int values

Radius and Angle can be double

Angle is a decimal value between 0 and $2\pi$

Each window has its own coordinate system