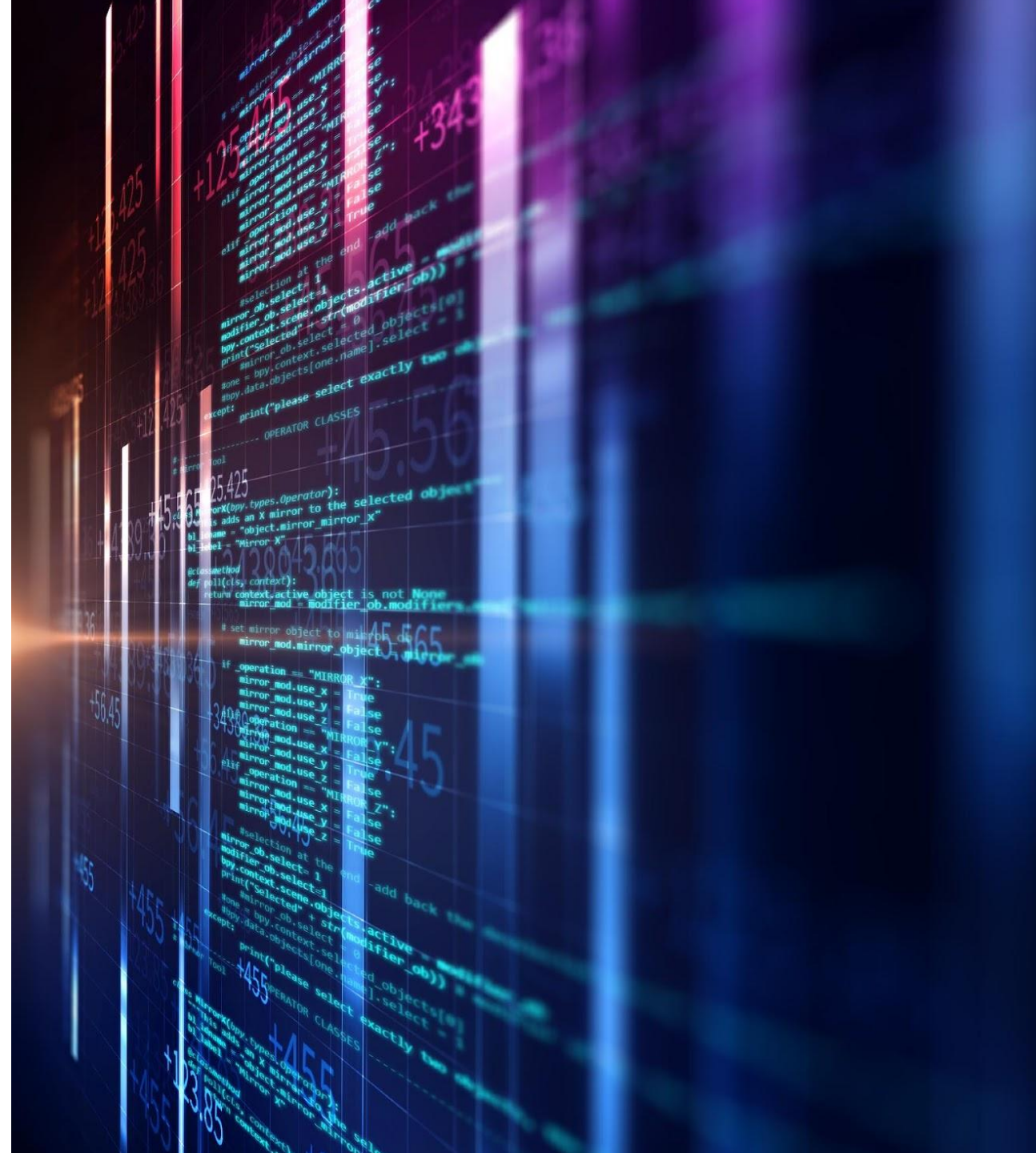# Polymorphism - Overloading

Brandon Krakowsky

Polymorphism

Signatures

In any programming language, a signature is what distinguishes one function or method from another

- In C, every function has to have a different name

- In Java, two methods have to differ in their *names* or in the *number* or *types* or *sequence* of their parameters

- For example:
    - foo(int i) and foo(int i, int j) are considered different
    - foo(int i) and foo(int k) are considered the same
    - foo(int i, double d) and foo(double d, int i) are considered different

- In Java, a method signature does not include the return type

Polymorphism

Polymorphism means *many* (poly) *shapes* (morph)

- In Java, polymorphism often refers to the fact that you can have multiple methods with the same *name* in the same *class*

- Polymorphism is divided into two types:
  - Overloading
    - Having two or more methods with the same *names* but different *signatures*
  - Overriding
    - Replacing an inherited method with another having the same *signature*

Overloading

Overloading

- Here's a class with 2 *myPrint* methods: they have different parameters

```java
public class MyPrintingUtility {

    //prints given int i
    public void myPrint(int i) {
        System.out.println("int i = " + i);
    }

    //prints given double d
    public void myPrint(double d) { //same name, different parameter
        System.out.println("double d = " + d);
    }

    public static void main(String args[]) {
        MyPrintingUtility printingUtility = new MyPrintingUtility();
        printingUtility.myPrint(5);
        printingUtility.myPrint(5.0); //call same method name with different argument type
    }
}
```

Why Overload a Method?

- So you can use the same names for methods that do essentially the same thing

- These all take a single argument and print it
    - `System.out.println(int)`
    - `System.out.println(double)`
    - `System.out.println(boolean)`
    - `System.out.println(String)`
    - etc.

- These all take 2 arguments and compare them
    - `assertEquals(int expected, int actual)`
    - `assertEquals(String expected, String actual)`
    - `assertEquals(Object expected, Object actual)`
    - etc.

Why Overload a Method?

- So you can supply defaults for the parameters:

```java
public class MyCountingUtility {

    int count = 0;

    //increments count by given amount
    //returns count
    public int increment(int amount) {
        this.count += amount;
        return this.count;
    }

    //increments by 1 and returns count
    public int increment() {
        return this.increment(1); //Note, one method can call another of the
same name
    }
}
```

Why Overload a Method?

- So you can supply additional information:

```java
public class MyResults {

    double total = 0.0;
    double average = 0.0;

    //prints total and average
    public void printResults() {
        System.out.println("total = " + this.total + ", average = " +
this.average);
    }

    //prints given message and prints results
    public void printResults(String message) {
        System.out.println(message + ": ");
        this.printResults();
    }
}
```

DRY (Don't Repeat Yourself) Principle of Software Development

- When you overload a method with another, very similar method, only one of them should do most of the work:

```java
public class MyInformation {
    int first;
    int last;
    String[] dictionary;
    //prints first, last, and dictionary info in between
    public void debug() {
        System.out.println("first = " + this.first + ", last = " + this.last);
        for (int i = this.first; i <= this.last; i++) {
            System.out.print(this.dictionary[i] + "  ");
        }
        System.out.println();
    }
    //prints given checkpoint s and debugs
    public void debug(String s) {
        System.out.println("At checkpoint " + s + ":");
        this.debug();
    }
}
```

Legal Variable Assignments

- In some cases, you can assign a different type of data to a predefined data type

- Widening (going to a "wider" data type) is legal
  `double d = 5; //legal`

- Narrowing (going to a more "narrow" data type) is illegal
  `int i = 3.5; //illegal`

  Unless you cast
  `int i = (int)(Math.round(3.5)); //legal`

- Rule: All ints are doubles but all doubles are not ints, so Java gets mad unless you do the cast!

Legal Method Calls

- Method calls have the same rules

- The following call to *myPrint* is legal due to widening

```
public class MyPrintingUtility {

    public void myPrint(double d) {
        System.out.println(d);
    }

    public static void main(String args[]) {
        MyPrintingUtility printingUtility = new MyPrintingUtility();
        printingUtility.myPrint(5); //widening is legal: will print 5.0
    }
}
```

Illegal Method Calls

- Method calls have the same rules

- The following call to *myPrint* is illegal due to narrowing

```java
public class MyPrintingUtility {

    public void myPrint(int i) {
        System.out.println(i);
    }

    public static void main(String args[]) {
        MyPrintingUtility printingUtility = new MyPrintingUtility();
        printingUtility.myPrint(5.0); //narrowing is illegal
    }
}
```

Java Uses the Most Specific Method

- If your methods are legally overloaded, Java will figure out which one you want to use

```java
public class MyPrintingUtility {

    public static void myPrint(double d) {
        System.out.println("double: " + d);
    }

    public static void myPrint(int i) {
        System.out.println("int: " + i);
    }

    public static void main(String args[]) {
        MyPrintingUtility.myPrint(5); //prints "int: 5" using myPrint(int i)
        MyPrintingUtility.myPrint(5.0); //prints "double: 5.0" using
myPrint(double d)
    }
}
```

Multiple Constructors I

- You can overload constructors as well as methods

```java
public class Counter {

    int count;

    //creates counter and starts count at 0
    public Counter() {
        this.count = 0;
    }

    //creates counter and starts count at given start
    public Counter(int start) {
        this.count = start;
    }
}
```

Multiple Constructors II

- One constructor can call another constructor in the same class, but there are rules
    - You call the other constructor with the keyword this
    - The call must be the *very first thing* the constructor does

```java
public class Point {

    int x;
    int y;
    int sum;

    //creates a point at given x and y
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
        this.sum = x + y;
    }

    //creates a point at 0, 0
    public Point() {
        this(0, 0);
    }
}
```

Summary

- Rule: You should *overload* a method when you want to do essentially the same thing, but with different parameters

Dog Project

Dog Class

```java
package poly;

/**
 * Represents a Dog.
 * @author lbrandon
 *
 */
public class Dog {

    //static variables

    /**
     * Default name for a dog.
     */
    static String DEFAULT_NAME = "Generic dog";

    /**
     * Default age for a dog.
     */
    static double DEFAULT_AGE = .5;

    /**
     * Default weight for a dog.
     */
    static double DEFAULT_WEIGHT = 1.75;

    /**
     * Default food for dog.
     */
    static String DEFAULT_FOOD = "Generic dog food";

    /**
     * Default dog bark sound.
     */
    static String DEFAULT_BARK = "Bark!";

    /**
     * Default number of times for dog to bark.
     */
    static int DEFAULT_NUM_TIMES_BARK = 1;

    /**
     * Constant weight gain value.
     */
    static final double WEIGHT_GAIN_INCREASE = 0.01;
```

Dog Class

```
46
47      //instance variables
48
49⊖     /**
50       * Name of dog.
51       */
52      String name;
53
54⊖     /**
55       * Age of dog.
56       */
57      double age;
58
59⊖     /**
60       * Owner of dog.
61       */
62      String owner;
63
64⊖     /**
65       * Weight of dog.
66       */
67      double weight;
68
```

Dog Class

```
68
69      //constructors
70
71⊖     /**
72       * Creates a Dog with given name, age, owner, and weight.
73       * @param name of dog
74       * @param age of dog
75       * @param owner for dog
76       * @param weight for dog
77       */
78⊖     public Dog(String name, double age, String owner, double weight) {
79          this.name = name;
80          this.age = age;
81          this.owner = owner;
82          this.weight = weight;
83      }
84
85⊖     /**
86       * Creates a Dog with given name and age.
87       * @param name of dog
88       * @param age of dog
89       */
90⊖     public Dog(String name, double age) {
91          //call first constructor with 2 given values
92          //and 2 default values
93          this(name, age, null, Dog.DEFAULT_WEIGHT);
94      }
95
96⊖     /**
97       * Creates a dog that barks immediately.
98       */
99⊖     public Dog() {
100         //call second constructor with 2 default values
101         this(Dog.DEFAULT_NAME, Dog.DEFAULT_AGE);
102
103         //dog barks after creation
104         this.bark();
105     }
106
```

Dog Class

```
106
107      /**
108       * Dog eats given amount of given food.
109       * @param amount to eat
110       * @param food to eat
111       * @return new weight
112       */
113      public double eat(double amount, String food) {
114          System.out.println(this.name + " is eating " + amount + " of " + food);
115
116          double weightGained = Dog.WEIGHT_GAIN_INCREASE * amount;
117
118          this.weight += weightGained;
119
120          return this.weight;
121      }
122
123      /**
124       * Dog eats given amount of generic dog food.
125       * @param amount to eat
126       * @return new weight
127       */
128      public double eat(double amount) {
129          //calls first eat method with given amount
130          //and default dog food
131          return this.eat(amount, Dog.DEFAULT_FOOD);
132      }
133
```

Dog Class

```java
        /**
         * Dog eats given amount.
         * Parses given amount as a double.
         * @param amount to eat
         * @return new weight
         */
        public double eat(String amount) {

            double returnVal = 0.0;

            //try some code in try block
            try {
                //cast given amount to double with static parseDouble method
                double amountAsDouble = Double.parseDouble(amount);

                //calls second eat method with given amount
                //and gets return value
                returnVal = this.eat(amountAsDouble);

            //code may fail, in which case we end up in catch block
            } catch (NumberFormatException e) {
                //print friendly message
                System.out.println(amount + ": can't be casted to double");
            }

            return returnVal;
        }
```

Dog Class

```java
162⊖    /**
163      * Dog makes given bark sound given number of times.
164      * @param numTimes to make sound
165      * @param barkSound to make
166      */
167⊖    public void bark(int numTimes, String barkSound) {
168          //prints dog's name
169          System.out.println(this.name + " says:");
170
171          //iterate using numTimes to print barkSound
172          for (int i = 0; i < numTimes; i++) {
173              System.out.println(barkSound);
174          }
175
176          System.out.println();
177      }
178
179⊖    /**
180      * Dog makes given bark sound given number of times.
181      * @param numTimes to make sound
182      * @param barkSound to make
183      */
184⊖    public void bark(String barkSound, int numTimes) {
185          //calls first bark method with given bark sound
186          //and number times
187          this.bark(numTimes, barkSound);
188      }
189
190⊖    /**
191      * Dog makes generic bark sound once.
192      */
193⊖    public void bark() {
194          //calls first bark method with default values
195          this.bark(Dog.DEFAULT_NUM_TIMES_BARK, Dog.DEFAULT_BARK);
196      }
197
```

Dog Class

```
198⊖    /**
199      * Returns dog's weight.
200      * @return weight
201      */
202⊖    public double getWeight() {
203          return this.weight;
204      }
205
206⊖    /**
207      * Set new name for dog.
208      * @param name of dog
209      */
210⊖    public void setName(String name) {
211          this.name = name;
212      }
213
214⊖    /**
215      * Sets dog's owner.
216      * @param owner for dog
217      */
218⊖    public void setOwner(String owner) {
219          this.owner = owner;
220      }
221
```

Dog Class

```
221
222    /**
223     * Returns String representing this dog.
224     */
225    @Override
226    public String toString() {
227        return this.name + ", " + this.weight + ", " + this.age + ", " + this.owner;
228    }
229
```

Dog Class

```java
public static void main(String[] args) {

    //create dog using first constructor
    Dog dog1 = new Dog("Princess", 12.7, "Brandon", 9.3);

    //create dog using second constructor
    Dog dog2 = new Dog("Fido", 5.5);

    //create dog using third constructor
    Dog dog3 = new Dog();

    //print dogs
    System.out.println(dog1);
    System.out.println(dog2);
    System.out.println(dog3);

    System.out.println("\n");
```

Dog Class

```
247
248            //set name for dog3
249            dog3.setName("Samantha");
250
251            //re-print dog3
252            System.out.println(dog3);
253
254            System.out.println("\n");
255
```

Dog Class

```java
255
256        //calls first eat method
257        //prints new weight
258        System.out.println(dog1.eat(2.1, "Beneful"));
259        System.out.println("\n");
260
261        //calls second eat method
262        System.out.println(dog2.eat(1.1));
263        System.out.println("\n");
264
265        //calls second eat method with int (widening)
266        System.out.println(dog3.eat(1));
267        System.out.println("\n");
268
269        //calls third eat method with string which can be parsed as a double
270        System.out.println(dog3.eat("12.1"));
271        System.out.println("\n");
272
273        //calls third eat method with string which cannot be parsed as a double
274        //should print friendly error
275        dog3.eat("twelve");
276
277        //print weight for dog3 -- it should be the same
278        System.out.println(dog3.getWeight());
279        System.out.println("\n");
280
```

Dog Class

```
281        //calls first bark method
282        dog1.bark(2, "Woof!");
283
284        //calls second bark method
285        dog3.bark("Help me!", 1);
286
287        //calls third default bark method
288        dog2.bark();
289    }
```