Data Structures and Algorithms in Java

Object-oriented Programming: Design Principles

Outline

Implementation Inheritance (Subclassing)

Implementation Inheritance (Subclassing)

Implementation inheritance allows us to define a new (sub) class that inherits from another (super) class

Implementation Inheritance (Subclassing)

Implementation inheritance allows us to define a new (sub) class that inherits from another (super) class

Typically, the subclass overrides some of the methods in the superclass

Shape

`double area()`	returns the area of this shape

Rectangle extends Shape

`Rectangle(double width, double height)`	constructs a rectangle given its width and height
`double area()`	returns the area of this rectangle
`String toString()`	returns a string representation of this rectangle

Circle extends Shape

`Circle(double radius)`	constructs a circle given its radius
`double area()`	returns the area of this circle
`String toString()`	returns a string representation of this circle

Implementation Inheritance (Subclassing) · Example (Shapes)

</> Shapes.java 1/2

```java
import stdlib.StdOut;

class Shape {
    public double area() {
        return Double.NaN;
    }
}

class Rectangle extends Shape {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    public double area() {
        return this.width * this.height;
    }

    public String toString() {
        return "Rectangle(w = " + this.width + ", h = " + this.height + ")";
    }
}

class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public double area() {
        return Math.PI * this.radius * this.radius;
```

Implementation Inheritance (Subclassing) · Example (Shapes)

</> Shapes.java 2/2

```
36      }
37
38      public String toString() {
39          return "Circle(r = " + this.radius + ")";
40      }
41  }
42
43  public class Shapes {
44      public static void main(String[] args) {
45          Rectangle r = new Rectangle(3, 4);
46          Circle c = new Circle(5);
47          StdOut.println("r        = " + r);
48          StdOut.println("r.area() = " + r.area());
49          StdOut.println("c        = " + c);
50          StdOut.println("c.area() = " + c.area());
51      }
52  }
```

Implementation Inheritance (Subclassing) · Example (Shapes)

```
>_ ~/workspace/dsaj/programs
$ _
```

```
>_ ~/workspace/dsaj/programs
$ java Shapes
```

`>_ ~/workspace/dsaj/programs`

```
$ java Shapes
r         = Rectangle(w = 3.0, h = 4.0)
r.area()  = 12.0
c         = Circle(r = 5.0)
c.area()  = 78.53981633974483
$ _
```

Interface Inheritance (Subtyping)

Interface Inheritance (Subtyping)

Interface inheritance allows us to declare relationships between otherwise unrelated classes

Interface Inheritance (Subtyping)

Interface inheritance allows us to declare relationships between otherwise unrelated classes

This is done using an interface that defines a contract of public methods that the classes must implement

Interface Inheritance (Subtyping)

Interface inheritance allows us to declare relationships between otherwise unrelated classes

This is done using an interface that defines a contract of public methods that the classes must implement

Interfaces don't provide implementations for their methods, but only declare their signatures

Interface Inheritance (Subtyping) · Example (Functions)

Function	
`double evaluate(double x)`	returns the value of the function at `x`

`Line implements Function`	
`Line()`	constructs an object representing the function $f(x) = x$
`double evaluate(double x)`	returns `x`

`Square implements Function`	
`Square()`	constructs an object representing the function $f(x) = x^2$
`double evaluate(double x)`	returns `x * x`

Interface Inheritance (Subtyping) · Example (Functions)

</> Functions.java

```java
import stdlib.StdOut;

interface Function {
    double evaluate(double x);
}

class Line implements Function {
    public double evaluate(double x) {
        return x;
    }
}

class Square implements Function {
    public double evaluate(double x) {
        return x * x;
    }
}

class Functions {
    public static void main(String[] args) {
        Function f = new Line();
        Function g = new Square();
        StdOut.println("Integral(x, 0, 1)   = " + integrate(f, 0, 1, 100));
        StdOut.println("Integral(x^2, 0, 1) = " + integrate(g, 0, 1, 100));
    }

    private static double integrate(Function f, double a, double b, int n) {
        double delta = (b - a) / n;
        double sum = 0.0;
        for (int i = 0; i < n; i++) {
            sum += delta * f.evaluate(a + delta * (i + 0.5));
        }
        return sum;
    }
}
```

Interface Inheritance (Subtyping) · Example (Functions)

```
>_ ~/workspace/dsaj/programs
$ _
```

Interface Inheritance (Subtyping) · Example (Functions)

```
>_ ~/workspace/dsaj/programs
$ java Functions
```

Interface Inheritance (Subtyping) · Example (Functions)

```
>_ ~/workspace/dsaj/programs

$ java Functions
Integral(x, 0, 1)   = 0.5
Integral(x^2, 0, 1) = 0.33332500000000004
$ _
```

java.lang.Iterable

`Iterator<Type> iterator()`	returns an iterator over a collection of items of type `Type`

java.util.Iterator

`boolean hasNext()`	returns `true` if the iterator has more items, and `false` otherwise
`Type next()`	returns the next item in the iterator

An `Iterable` object `o` can be iterated over using the for-each statement

```
for (T item : o) {
    <statement>
    ...
}
```

which is equivalent to

```
Iterator iter = o.iterator();
while (iter.hasNext()) {
    T item = iter.next();
    <statement>
    ...
}
```

Arrays are iterable, and thus can be iterated using the for-each statement

Arrays are iterable, and thus can be iterated using the for-each statement

Example

```
String[] dow = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};
for (String s : dow) {
    StdOut.println(s);
}
```

</> IterableClass.java

```
import java.util.Iterator;

public class IterableClass implements Iterable<T> {
    ...
    public Iterator<T> iterator() {
        return new AnIterator();
    }

    private class AnIterator implements Iterator<T> {
        ...
        public boolean hasNext() {
            ...
        }

        public T next() {
            ...
        }
    }
    ...
}
```

Iterable Classes · Example (`Words` Class)

`Words implements Iterable`	
`Words(String sentence)`	constructs a `Words` object from the given sentence
`String toString()`	returns a string representation of the words in the sentence
`Iterator<String> iterator()`	returns an iterator to iterate over the words in the sentence

Iterable Classes · Example (`Words` Class)

`Words` implements `Iterable`	
`Words(String sentence)`	constructs a `Words` object from the given sentence
`String toString()`	returns a string representation of the words in the sentence
`Iterator<String> iterator()`	returns an iterator to iterate over the words in the sentence

Instance variables

- Words in the sentence, `String[] words`

Iterable Classes · Example (`Words` Class)

Words.java

Command-line input	*sentence* (String)
Standard output	the words in *sentence*

>_ ~/workspace/dsaj/programs

```
$ _
```

Iterable Classes · Example (`Words` Class)

Words.java

Command-line input	*sentence* (String)
Standard output	the words in *sentence*

>_ ~/workspace/dsaj/programs

```
$ java Words "it was the best of times it was the worst of times"
```

Iterable Classes · Example (`Words` Class)

Words.java

Command-line input	*sentence* (String)
Standard output	the words in *sentence*

>_ ~/workspace/dsaj/programs

```
$ java Words "it was the best of times it was the worst of times"
it was the best of times it was the worst of times
$ _
```

Iterable Classes · Example (Words Class)

</> Words.java 1/2

```java
import java.util.Iterator;
import stdlib.StdOut;

public class Words implements Iterable<String> {
    private String[] words;

    public Words(String sentence) {
        this.words = sentence.split("\\s+");
    }

    public String toString() {
        String s = "";
        for (String v : this) {
            s += v + " ";
        }
        return s.trim();
    }

    public Iterator<String> iterator() {
        return new WordsIterator();
    }

    private class WordsIterator implements Iterator<String> {
        private int current;

        public WordsIterator() {
            this.current = 0;
        }

        public boolean hasNext() {
            return this.current < words.length;
        }

        public String next() {
            return words[this.current++];
```

Iterable Classes · Example (Words Class)

Words.java 2/2

```
36          }
37      }
38
39      public static void main(String[] args) {
40          String sentence = args[0];
41          Words words = new Words(sentence);
42          StdOut.println(words);
43      }
44  }
```

Iterable Classes · Example (`FibonacciSequence` Class)

`FibonacciSequence implements Iterable`

`FibonacciSequence(int n)`	constructs a `FibonacciSequence` object given the length of the sequence
`String toString()`	returns a string representation of the object
`Iterator<String> iterator()`	returns an iterator to iterate over the numbers in the Fibonacci sequence

Iterable Classes · Example (`FibonacciSequence` Class)

`FibonacciSequence implements Iterable`	
`FibonacciSequence(int n)`	constructs a `FibonacciSequence` object given the length of the sequence
`String toString()`	returns a string representation of the object
`Iterator<String> iterator()`	returns an iterator to iterate over the numbers in the Fibonacci sequence

Instance variables

- Length of the Fibonacci sequence, `int n`

Iterable Classes · Example (`FibonacciSequence` Class)

`FibonacciSequence.java`

Command-line input	*n* (int)
Standard output	the first *n* numbers in the Fibonacci sequence

```
>_ ~/workspace/dsaj/programs

$ _
```

`FibonacciSequence.java`

Command-line input	*n* (int)
Standard output	the first *n* numbers in the Fibonacci sequence

`~/workspace/dsaj/programs`

```
$ java FibonacciSequence 10
```

FibonacciSequence.java

Command-line input	*n* (int)
Standard output	the first *n* numbers in the Fibonacci sequence

>_ ~/workspace/dsaj/programs

```
$ java FibonacciSequence 10
0 1 1 2 3 5 8 13 21 34
$ _
```

Iterable Classes · Example (FibonacciSequence Class)

</> FibonacciSequence.java 1/2

```java
import java.util.Iterator;
import stdlib.StdOut;

public class FibonacciSequence implements Iterable<Long> {
    private int n;

    public FibonacciSequence(int n) {
        this.n = n;
    }

    public String toString() {
        String s = "";
        for (long v : this) {
            s += v + " ";
        }
        return s.trim();
    }

    public Iterator<Long> iterator() {
        return new FibonacciIterator();
    }

    private class FibonacciIterator implements Iterator<Long> {
        private long a;
        private long b;
        private int count;

        public FibonacciIterator() {
            this.a = -1;
            this.b = 1;
            this.count = 0;
        }

        public boolean hasNext() {
            return this.count < n;
```

Iterable Classes · Example (FibonacciSequence Class)

</> FibonacciSequence.java 2/2

```
        }

        public Long next() {
            this.count++;
            long temp = this.a;
            this.a = this.b;
            this.b += temp;
            return this.b;
        }
    }

    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        FibonacciSequence fib = new FibonacciSequence(n);
        StdOut.println(fib);
    }
}
```

java.lang.Comparable

`int compareTo(Type other)`	returns a comparison of this object with `other`

java.util.Comparator

`int compare(Type v, Type w)`	returns a comparison of object `v` with object `w`

Comparable Classes · Comparable Class Template

</> ComparableClass.java

```java
import java.util.Comparator;

public class ComparableClass implements Comparable<ComparableClass> {
    ...
    // Natural ordering.
    public int compareTo(ComparableClass other) {
        ...
    }

    public static Comparator<ComparableClass> aOrder() {
        return new AOrder();
    }

    public static Comparator<ComparableClass> bOrder() {
        return new BOrder();
    }

    // Alternate ordering 1.
    private static class AOrder implements Comparator<ComparableClass> {
        ...
        public int compare(ComparableClass v, ComparableClass w) {
            ...
        }
    }

    // Alternate ordering 2.
    private static class BOrder implements Comparator<ComparableClass> {
        ...
        public int compare(ComparableClass v, ComparableClass w) {
            ...
        }
    }
    ...
}
```

Comparable Classes · Example (`Counter` Class)

`dsa.Counter implements java.lang.Comparable<Counter>`

`Counter(String id)`	constructs a counter given its id
`void increment()`	increments the counter by 1
`int tally()`	returns the current value of the counter
`void reset()`	resets the counter to zero
`boolean equals(Object other)`	returns `true` if the counter and `other` have the same tally, and `false` otherwise
`String toString()`	returns a string representation of the counter
`int compareTo(Counter other)`	returns a comparison of the counter with `other` based on their tally

Comparable Classes · Example (`Counter` Class)

`dsa.Counter implements java.lang.Comparable<Counter>`

`Counter(String id)`	constructs a counter given its id
`void increment()`	increments the counter by 1
`int tally()`	returns the current value of the counter
`void reset()`	resets the counter to zero
`boolean equals(Object other)`	returns `true` if the counter and `other` have the same tally, and `false` otherwise
`String toString()`	returns a string representation of the counter
`int compareTo(Counter other)`	returns a comparison of the counter with `other` based on their tally

Instance variables

- Counter ID, `String id`
- Counter tally, `int tally`

`Counter.java`

Command-line input	*n* (int) and *trials* (int)
Standard output	face value frequencies of an *n*-sided die rolled *trials* number of times

Counter.java

Command-line input	*n* (int) and *trials* (int)
Standard output	face value frequencies of an *n*-sided die rolled *trials* number of times

```
>_ ~/workspace/dsaj/programs
$ _
```

`Counter.java`

Command-line input	*n* (int) and *trials* (int)
Standard output	face value frequencies of an *n*-sided die rolled *trials* number of times

`>_ ~/workspace/dsaj/programs`

```
$ java dsa.Counter
```

Comparable Classes · Example (`Counter` Class)

Counter.java

Command-line input	*n* (int) and *trials* (int)
Standard output	face value frequencies of an *n*-sided die rolled *trials* number of times

>_ ~/workspace/dsaj/programs

```
$ java dsa.Counter
513 counter 0
487 counter 1
$ _
```

Comparable Classes · Example (Counter Class)

</> Counter.java 1/2

```java
package dsa;

import stdlib.StdOut;
import stdlib.StdRandom;

public class Counter implements Comparable<Counter> {
    private String id;
    private int count;

    public Counter(String id) {
        this.id = id;
        this.count = 0;
    }

    public void increment() {
        this.count++;
    }

    public int tally() {
        return this.count;
    }

    public void reset() {
        this.count = 0;
    }

    public boolean equals(Object other) {
        if (other == null) {
            return false;
        }
        if (other == this) {
            return true;
        }
        if (other.getClass() != this.getClass()) {
            return false;
```

Comparable Classes · Example (Counter Class)

Counter.java 2/2

```
        }
        Counter a = this, b = (Counter) other;
        return a.count == b.count;
    }

    public String toString() {
        return this.count + " " + this.id;
    }

    public int compareTo(Counter other) {
        return Integer.compare(this.count, other.count);
    }

    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int trials = Integer.parseInt(args[1]);
        Counter[] hits = new Counter[n];
        for (int i = 0; i < n; i++) {
            hits[i] = new Counter("counter " + i);
        }
        for (int t = 0; t < trials; t++) {
            hits[StdRandom.uniform(n)].increment();
        }
        for (int i = 0; i < n; i++) {
            StdOut.println(hits[i]);
        }
    }
}
```

Comparable Classes · Example (`Transaction` Class)

`dsa.Transaction implements java.lang.Comparable<Transaction>`

`Transaction(String name, Date date, double amount)`	constructs a transaction from a `name`, `date`, and `amount`
`Transaction(String s)`	constructs a transaction from a string `s` of the form "name date amount"
`String name()`	returns the name of the person involved in this transaction
`Date date()`	returns the date of this transaction
`double amount()`	returns the amount of this transaction
`int hashCode()`	returns a hash code for this transaction
`String toString()`	returns a string representation of this transaction
`int compareTo(Transaction other)`	returns a comparison of this transaction with `other` by amount
`static Comparator<Transaction> nameOrder()`	returns a comparator for comparing two transactions by name
`static Comparator<Transaction> dateOrder()`	returns a comparator for comparing two transactions by date

Comparable Classes · Example (`Transaction` Class)

`dsa.Transaction implements java.lang.Comparable<Transaction>`

`Transaction(String name, Date date, double amount)`	constructs a transaction from a `name`, `date`, and `amount`
`Transaction(String s)`	constructs a transaction from a string `s` of the form "name date amount"
`String name()`	returns the name of the person involved in this transaction
`Date date()`	returns the date of this transaction
`double amount()`	returns the amount of this transaction
`int hashCode()`	returns a hash code for this transaction
`String toString()`	returns a string representation of this transaction
`int compareTo(Transaction other)`	returns a comparison of this transaction with `other` by amount
`static Comparator<Transaction> nameOrder()`	returns a comparator for comparing two transactions by name
`static Comparator<Transaction> dateOrder()`	returns a comparator for comparing two transactions by date

Instance variables

- Transaction name, `String name`
- Transaction date, `Date date`
- Transaction amount, `double amount`

`Transaction.java`

Standard output	four transactions (one per line) in different orders

Comparable Classes · Example (`Transaction` Class)

Transaction.java

Standard output | four transactions (one per line) in different orders

```
>_ ~/workspace/dsaj/programs
$ _
```

Comparable Classes · Example (`Transaction` Class)

Transaction.java

Standard output	four transactions (one per line) in different orders

>_ ~/workspace/dsaj/programs

```
$ java dsa.Transaction
```

Comparable Classes · Example (`Transaction` Class)

Transaction.java

Standard output	four transactions (one per line) in different orders

~/workspace/dsaj/programs

```
$ java dsa.Transaction
Unsorted:
Turing      6/17/1990   644.08
Tarjan      3/26/2002  4121.85
Knuth       6/14/1999   288.34
Dijkstra    8/22/2007  2678.40

Sorted by name:
Dijkstra    8/22/2007  2678.40
Knuth       6/14/1999   288.34
Tarjan      3/26/2002  4121.85
Turing      6/17/1990   644.08

Sorted by date:
Turing      6/17/1990   644.08
Knuth       6/14/1999   288.34
Tarjan      3/26/2002  4121.85
Dijkstra    8/22/2007  2678.40

Sorted by amount:
Knuth       6/14/1999   288.34
Turing      6/17/1990   644.08
Dijkstra    8/22/2007  2678.40
Tarjan      3/26/2002  4121.85
$ _
```

Comparable Classes · Example (Transaction Class)

</> Transaction.java 1/3

```java
package dsa;

import java.util.Comparator;
import stdlib.StdOut;

public class Transaction implements Comparable<Transaction> {
    private String name;
    private Date date;
    private double amount;

    public Transaction(String name, Date date, double amount) {
        this.name = name;
        this.date = date;
        this.amount = amount;
    }

    public Transaction(String s) {
        String[] a = s.split("\\s+");
        this.name = a[0];
        this.date = new Date(a[1]);
        this.amount = Double.parseDouble(a[2]);
    }

    public String name() {
        return this.name;
    }

    public Date date() {
        return this.date;
    }

    public double amount() {
        return this.amount;
    }

```

Comparable Classes · Example (Transaction Class)

Transaction.java 2/3

```
36      public int hashCode() {
37          int hash = 1;
38          hash = 31 * hash + this.name.hashCode();
39          hash = 31 * hash + this.date.hashCode();
40          hash = 31 * hash + ((Double) this.amount).hashCode();
41          return hash;
42      }
43
44      public String toString() {
45          return String.format("%-10s %10s %8.2f", this.name, this.date, this.amount);
46      }
47
48      public int compareTo(Transaction other) {
49          return Double.compare(this.amount, other.amount);
50      }
51
52      public static Comparator<Transaction> nameOrder() {
53          return new NameOrder();
54      }
55
56      public static Comparator<Transaction> dateOrder() {
57          return new DateOrder();
58      }
59
60      private static class NameOrder implements Comparator<Transaction> {
61          public int compare(Transaction v, Transaction w) {
62              return v.name.compareTo(w.name);
63          }
64      }
65
66      private static class DateOrder implements Comparator<Transaction> {
67          public int compare(Transaction v, Transaction w) {
68              return v.date.compareTo(w.date);
69          }
70      }
```

Comparable Classes · Example (Transaction Class)

</> Transaction.java 3/3

```
71
72      public static void main(String[] args) {
73          Transaction[] transactions = new Transaction[4];
74          transactions[0] = new Transaction("Turing   6/17/1990  644.08");
75          transactions[1] = new Transaction("Tarjan   3/26/2002 4121.85");
76          transactions[2] = new Transaction("Knuth    6/14/1999  288.34");
77          transactions[3] = new Transaction("Dijkstra 8/22/2007 2678.40");
78          StdOut.println("Unsorted:");
79          for (Transaction transaction : transactions) {
80              StdOut.println(transaction);
81          }
82          StdOut.println();
83          StdOut.println("Sorted by name:");
84          Quick.sort(transactions, Transaction.nameOrder());
85          for (Transaction transaction : transactions) {
86              StdOut.println(transaction);
87          }
88          StdOut.println();
89          StdOut.println("Sorted by date:");
90          Quick.sort(transactions, Transaction.dateOrder());
91          for (Transaction transaction : transactions) {
92              StdOut.println(transaction);
93          }
94          StdOut.println();
95          StdOut.println("Sorted by amount:");
96          Quick.sort(transactions);
97          for (Transaction transaction : transactions) {
98              StdOut.println(transaction);
99          }
100     }
101 }
```

Error Handling

Error Handling

Errors (aka exceptions) are disruptive events that occur while a program is running

Error Handling

Errors (aka exceptions) are disruptive events that occur while a program is running

Example: `ArrayIndexOutOfBoundsException` and `NullPointerException`

Error Handling

Errors (aka exceptions) are disruptive events that occur while a program is running

Example: `ArrayIndexOutOfBoundsException` and `NullPointerException`

Throwing an exception

```
throw new <exception>(<message>);
```

Error Handling

Errors (aka exceptions) are disruptive events that occur while a program is running

Example: `ArrayIndexOutOfBoundsException` and `NullPointerException`

Throwing an exception

```
throw new <exception>(<message>);
```

Example

```
throw new IllegalArgumentException("x must be positive");
```

Error Handling

Error Handling

Catching an exception

```
try {
    <statement>
    ...
}
catch (<exception> e) {
    <statement>
    ...
}
catch (<exception> e) {
    <statement>
    ...
}
...
finally {
    <statement>
    ...
}
...
```

Flips.java

Command-line input	x (int)
Standard output	$\sqrt{x}$

Flips.java

Command-line input	x (int)
Standard output	$\sqrt{x}$

>_ ~/workspace/dsaj/programs

```
$ _
```

Flips.java

Command-line input	x (int)
Standard output	$\sqrt{x}$

~/workspace/dsaj/programs

```
$ java ErrorHandling
```

Error Handling · Example (Square Root)

Flips.java

Command-line input	x (int)
Standard output	$\sqrt{x}$

~/workspace/dsaj/programs

```
$ java ErrorHandling
x not specified
Done!
$ _
```

Error Handling · Example (Square Root)

Flips.java

Command-line input	x (int)
Standard output	$\sqrt{x}$

~/workspace/dsaj/programs

```
$ java ErrorHandling
x not specified
Done!
$ java ErrorHandling two
```

Error Handling · Example (Square Root)

Flips.java

Command-line input	x (int)
Standard output	$\sqrt{x}$

~/workspace/dsaj/programs

```
$ java ErrorHandling
x not specified
Done!
$ java ErrorHandling two
x must be a double
Done!
$ _
```

Error Handling · Example (Square Root)

Flips.java

Command-line input	x (int)
Standard output	$\sqrt{x}$

~/workspace/dsaj/programs

```
$ java ErrorHandling
x not specified
Done!
$ java ErrorHandling two
x must be a double
Done!
$ java ErrorHandling -2
```

Error Handling · Example (Square Root)

Flips.java

Command-line input	x (int)
Standard output	$\sqrt{x}$

~/workspace/dsaj/programs

```
$ java ErrorHandling
x not specified
Done!
$ java ErrorHandling two
x must be a double
Done!
$ java ErrorHandling -2
x must be positive
Done!
$ _
```

Error Handling · Example (Square Root)

Flips.java

Command-line input	x (int)
Standard output	$\sqrt{x}$

~/workspace/dsaj/programs

```
$ java ErrorHandling
x not specified
Done!
$ java ErrorHandling two
x must be a double
Done!
$ java ErrorHandling -2
x must be positive
Done!
$ java ErrorHandling 2
```

Error Handling · Example (Square Root)

Flips.java

Command-line input	x (int)
Standard output	$\sqrt{x}$

>_ ~/workspace/dsaj/programs

```
$ java ErrorHandling
x not specified
Done!
$ java ErrorHandling two
x must be a double
Done!
$ java ErrorHandling -2
x must be positive
Done!
$ java ErrorHandling 2
1.4142135623730951
Done!
$ _
```

Error Handling · Example (Square Root)

</> ErrorHandling.java

```java
import stdlib.StdOut;

public class ErrorHandling {
    public static void main(String[] args) {
        try {
            double x = Double.parseDouble(args[0]);
            double result = sqrt(x);
            StdOut.println(result);
        } catch (ArrayIndexOutOfBoundsException e) {
            StdOut.println("x not specified");
        } catch (NumberFormatException e) {
            StdOut.println("x must be a double");
        } catch (IllegalArgumentException e) {
            StdOut.println(e.getMessage());
        } finally {
            StdOut.println("Done!");
        }
    }

    private static double sqrt(double x) {
        if (x < 0) {
            throw new IllegalArgumentException("x must be positve");
        }
        return Math.sqrt(x);
    }
}
```