Data Abstraction

Outline

1. Abstract Data Type (ADT)

2. Using an ADT

3. Examples of ADTs

4. Defining an ADT

5. Error Handling

Abstract Data Type (ADT)

Abstract Data Type (ADT)

An abstract data type (ADT) is one whose representation is hidden from the program that uses the ADT

Abstract Data Type (ADT)

An abstract data type (ADT) is one whose representation is hidden from the program that uses the ADT

Example

`Counter implements Comparable<Counter>`

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

Abstract Data Type (ADT)

Abstract Data Type (ADT)

Salient features of an ADT:

Abstract Data Type (ADT)

Salient features of an ADT:

- ⇝ Some entries (called constructors) have the same name as the class and no return type

Abstract Data Type (ADT)

Salient features of an ADT:

- ⇝ Some entries (called constructors) have the same name as the class and no return type
- ⇝ Some entries (called methods) lack the `static` keyword and operate on data-type values

Abstract Data Type (ADT)

Salient features of an ADT:

- ⇝ Some entries (called constructors) have the same name as the class and no return type
- ⇝ Some entries (called methods) lack the `static` keyword and operate on data-type values
- ⇝ Some methods such as `equals()`, `hashCode()`, and `toString()` are inherited from the parent `java.lang.Object` class and overridden in the ADT

Using an ADT

Using an ADT

An object is an entity that can take on a data-type value

Using an ADT

An object is an entity that can take on a data-type value

Creating an object

```
<type> <name> = new <type>(<argument1>, <argument2>, ...);
```

Using an ADT

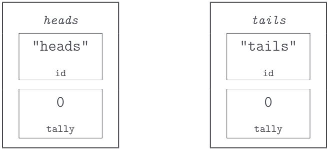
An object is an entity that can take on a data-type value

Creating an object

```
<type> <name> = new <type>(<argument1>, <argument2>, ...);
```

Example

```
Counter heads = new Counter("heads");
Counter tails = new Counter("tails");
```



Using an ADT

Using an ADT

A method, invoked as `[<object>].<name>(<argument1>, <argument2>, ...)`, operates on data-type values

Using an ADT

A method, invoked as `[<object>].<name>(<argument1>, <argument2>, ...)`, operates on data-type values

Example

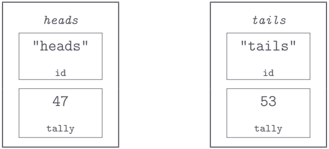
```
for (int i = 0; i < 100; i++) {
    if (StdRandom.bernoulli(0.5)) {
        heads.increment();
    } else {
        tails.increment();
    }
}
```

Using an ADT

A method, invoked as `[<object>].<name>(<argument1>, <argument2>, ...)`, operates on data-type values

Example

```java
for (int i = 0; i < 100; i++) {
    if (StdRandom.bernoulli(0.5)) {
        heads.increment();
    } else {
        tails.increment();
    }
}
```



heads

"heads"

id

47

tally

tails

"tails"

id

53

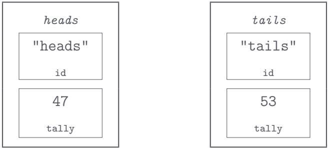
tally

Using an ADT

A method, invoked as `[<object>].<name>(<argument1>, <argument2>, ...)`, operates on data-type values

Example

```
for (int i = 0; i < 100; i++) {
    if (StdRandom.bernoulli(0.5)) {
        heads.increment();
    } else {
        tails.increment();
    }
}
```



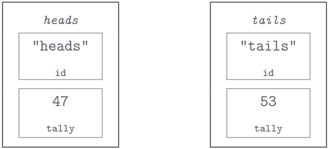
```
StdOut.println(heads.tally());
StdOut.println(tails.tally());
```

Using an ADT

A method, invoked as `[<object>].<name>(<argument1>, <argument2>, ...)`, operates on data-type values

Example

```java
for (int i = 0; i < 100; i++) {
    if (StdRandom.bernoulli(0.5)) {
        heads.increment();
    } else {
        tails.increment();
    }
}
```



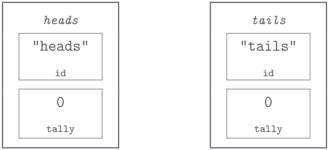
```java
StdOut.println(heads.tally());
StdOut.println(tails.tally());
```

```
47
53
```

Using an ADT

Using an ADT

```java
Counter heads = new Counter("heads");
Counter tails = new Counter("tails");
```



heads

"heads"

id

0

tally

tails

"tails"

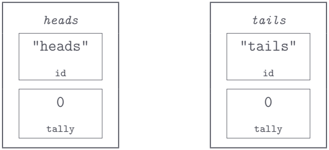
id

0

tally

Using an ADT

```java
Counter heads = new Counter("heads");
Counter tails = new Counter("tails");
```



heads

"heads"
id

0
tally

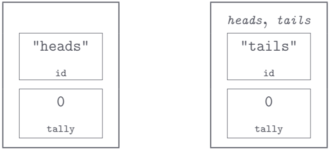
tails

"tails"
id

0
tally

Aliasing

```java
heads = tails;
```



"heads"
id

0
tally

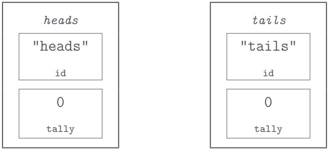
heads, *tails*

"tails"
id

0
tally

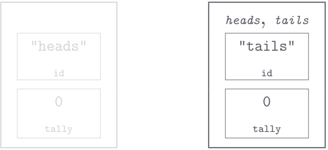
Using an ADT

```
Counter heads = new Counter("heads");
Counter tails = new Counter("tails");
```



Aliasing

```
heads = tails;
```



Using an ADT

Using an ADT

Two objects `x` and `y` must be compared for equality as `x.equals(y)` and not as `x == y`

Using an ADT

Two objects `x` and `y` must be compared for equality as `x.equals(y)` and not as `x == y`

Example

```java
String x = "Hello, World";
String y = "Hello, World";
String z = "Cogito, ergo sum";
StdOut.println("x == x? " + (x == x));
StdOut.println("x == y? " + (x == y));
StdOut.println("x == z? " + (x == z));
StdOut.println("x.equals(x)? " + x.equals(x));
StdOut.println("x.equals(y)? " + x.equals(y));
StdOut.println("x.equals(z)? " + x.equals(z));
```

Using an ADT

Two objects `x` and `y` must be compared for equality as `x.equals(y)` and not as `x == y`

Example

```
String x = "Hello, World";
String y = "Hello, World";
String z = "Cogito, ergo sum";
StdOut.println("x == x? " + (x == x));
StdOut.println("x == y? " + (x == y));
StdOut.println("x == z? " + (x == z));
StdOut.println("x.equals(x)? " + x.equals(x));
StdOut.println("x.equals(y)? " + x.equals(y));
StdOut.println("x.equals(z)? " + x.equals(z));
```

```
x == x? true
x == y? false
x == z? false
x.equals(x)? true
x.equals(y)? true
x.equals(z)? false
```

Using an ADT

Using an ADT

Program: `Flips.java`

Using an ADT

Program: `Flips.java`

$\rightsquigarrow$ Command-line input: n (int)

Using an ADT

Program: `Flips.java`

- ⇝ Command-line input: n (int)
- ⇝ Standard output: number of heads, tails, and the difference from n coin flips

Using an ADT

Program: `Flips.java`

- Command-line input: n (int)
- Standard output: number of heads, tails, and the difference from n coin flips

```
>_ ~/workspace/dsa/programs

$ _
```

Using an ADT

Program: `Flips.java`

- Command-line input: n (int)
- Standard output: number of heads, tails, and the difference from n coin flips

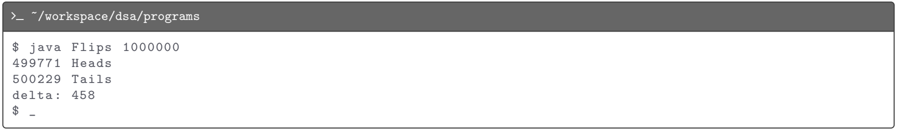
```
>_ ~/workspace/dsa/programs

$ java Flips 1000000
```

Using an ADT

Program: `Flips.java`

- ⇝ Command-line input: n (int)
- ⇝ Standard output: number of heads, tails, and the difference from n coin flips



>_ ~/workspace/dsa/programs

```
$ java Flips 1000000
499771 Heads
500229 Tails
delta: 458
$ _
```

Using an ADT

Using an ADT

Flips.java

```java
import dsa.Counter;
import stdlib.StdOut;
import stdlib.StdRandom;

public class Flips {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        Counter heads = new Counter("Heads");
        Counter tails = new Counter("Tails");
        for (int i = 0; i < n; i++) {
            if (StdRandom.bernoulli(0.5)) {
                heads.increment();
            } else {
                tails.increment();
            }
        }
        StdOut.println(heads);
        StdOut.println(tails);
        StdOut.println("delta: " + Math.abs(heads.tally() - tails.tally()));
    }
}
```

Using an ADT

Using an ADT

Program: `FlipsMax.java`

Using an ADT

Program: `FlipsMax.java`

⇝ Command-line input: n (int)

Using an ADT

Program: `FlipsMax.java`

- ⇝ Command-line input: n (int)
- ⇝ Standard output: the winner from n coin flips

Using an ADT

Program: `FlipsMax.java`

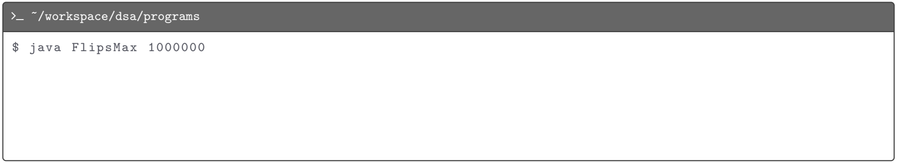
- Command-line input: n (int)
- Standard output: the winner from n coin flips

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

Using an ADT

Program: `FlipsMax.java`

- ⇝ Command-line input: n (int)
- ⇝ Standard output: the winner from n coin flips



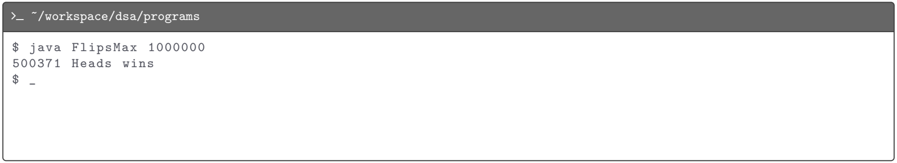
```
>_ ~/workspace/dsa/programs
```

```
$ java FlipsMax 1000000
```

Using an ADT

Program: `FlipsMax.java`

- ⇝ Command-line input: n (int)
- ⇝ Standard output: the winner from n coin flips



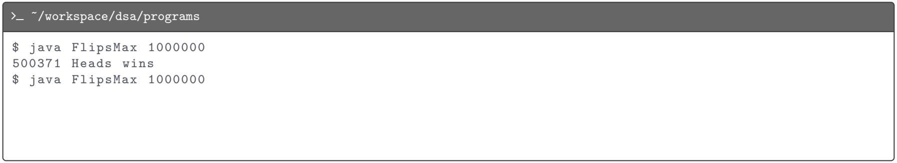
```
>_ ~/workspace/dsa/programs
```

```
$ java FlipsMax 1000000
500371 Heads wins
$ _
```

Using an ADT

Program: `FlipsMax.java`

- ⇝ Command-line input: n (int)
- ⇝ Standard output: the winner from n coin flips



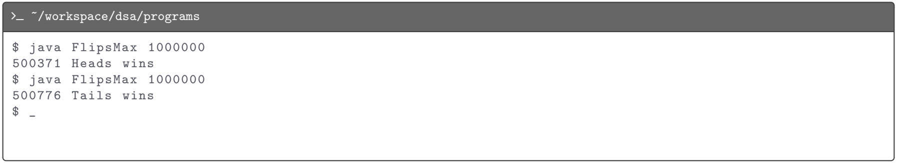
```
>_ ~/workspace/dsa/programs
```

```
$ java FlipsMax 1000000
500371 Heads wins
$ java FlipsMax 1000000
```

Using an ADT

Program: `FlipsMax.java`

- ⇝ Command-line input: n (int)
- ⇝ Standard output: the winner from n coin flips



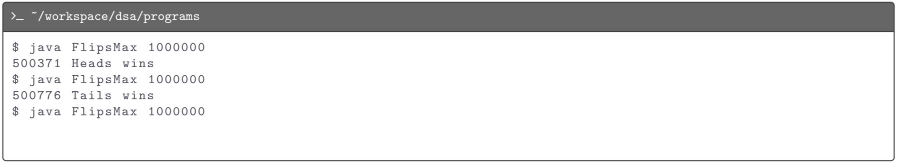
```
>_ ~/workspace/dsa/programs
```

```
$ java FlipsMax 1000000
500371 Heads wins
$ java FlipsMax 1000000
500776 Tails wins
$ _
```

Using an ADT

Program: `FlipsMax.java`

- ⇝ Command-line input: n (int)
- ⇝ Standard output: the winner from n coin flips



```
>_ ~/workspace/dsa/programs
```

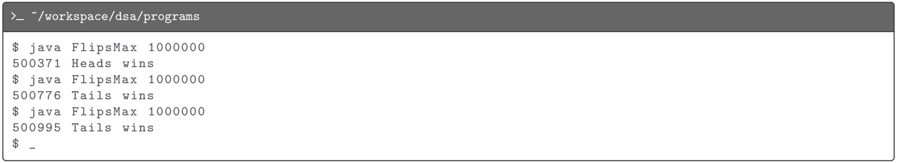
```
$ java FlipsMax 1000000
500371 Heads wins
$ java FlipsMax 1000000
500776 Tails wins
$ java FlipsMax 1000000
```

Using an ADT

Program: `FlipsMax.java`

⇝ Command-line input: n (int)

⇝ Standard output: the winner from n coin flips



```
$ java FlipsMax 1000000
500371 Heads wins
$ java FlipsMax 1000000
500776 Tails wins
$ java FlipsMax 1000000
500995 Tails wins
$ _
```

Using an ADT

Using an ADT

FlipsMax.java

```java
import dsa.Counter;
import stdlib.StdOut;
import stdlib.StdRandom;

public class FlipsMax {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        Counter heads = new Counter("Heads");
        Counter tails = new Counter("Tails");
        for (int i = 0; i < n; i++) {
            if (StdRandom.bernoulli(0.5)) {
                heads.increment();
            } else {
                tails.increment();
            }
        }
        if (heads.equals(tails)) {
            StdOut.println("Tie");
        } else {
            StdOut.println(max(heads, tails) + " wins");
        }
    }

    private static Counter max(Counter x, Counter y) {
        if (x.tally() > y.tally()) {
            return x;
        }
        return y;
    }
}
```

Using an ADT

Using an ADT

Program: `Rolls.java`

Using an ADT

Program: `Rolls.java`

$\rightsquigarrow$ Command-line input: n (int)

Using an ADT

Program: `Rolls.java`

- ⇝ Command-line input: n (int)
- ⇝ Standard output: frequencies of face values from rolling n 6-sided dice

Using an ADT

Program: `Rolls.java`

- ⇝ Command-line input: n (int)
- ⇝ Standard output: frequencies of face values from rolling n 6-sided dice

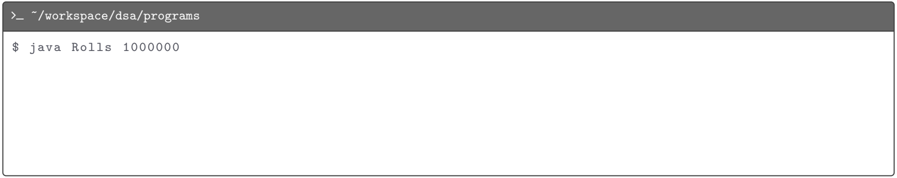
```
>_ ~/workspace/dsa/programs
```

```
$ _
```

Using an ADT

Program: `Rolls.java`

- ⇝ Command-line input: n (int)
- ⇝ Standard output: frequencies of face values from rolling n 6-sided dice



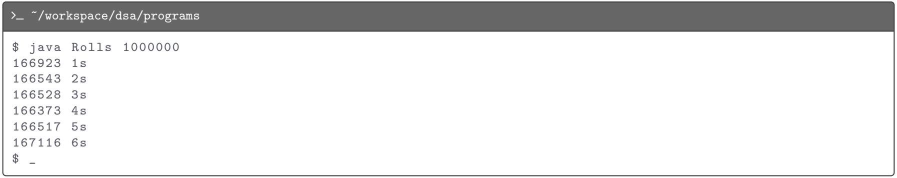
```
$ java Rolls 1000000
```

Using an ADT

Program: `Rolls.java`

⇝ Command-line input: n (int)

⇝ Standard output: frequencies of face values from rolling n 6-sided dice



```
$ java Rolls 1000000
166923 1s
166543 2s
166528 3s
166373 4s
166517 5s
167116 6s
$ _
```

Using an ADT

Using an ADT

Rolls.java

```java
import dsa.Counter;
import stdlib.StdOut;
import stdlib.StdRandom;

public class Rolls {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int SIDES = 6;
        Counter[] rolls = new Counter[SIDES + 1];
        for (int i = 1; i <= SIDES; i++) {
            rolls[i] = new Counter(i + "s");
        }
        for (int j = 0; j < n; j++) {
            int result = StdRandom.uniform(1, SIDES + 1);
            rolls[result].increment();
        }
        for (int i = 1; i <= SIDES; i++) {
            StdOut.println(rolls[i]);
        }
    }
```

Examples of ADTs

Examples of ADTs

Type	Description
`java.lang.Integer`	wraps a primitive `int`
`java.lang.Double`	wraps a primitive `double`
`java.lang.String`	represents a sequence of characters
`java.util.NoSuchElementException`	used to indicate that the requested element does not exist
`dsa.Stopwatch`	represents a stopwatch
`dsa.WeightedQuickUnionUF`	represents the union-find data structure

Examples of ADTs · String

Examples of ADTs · String

☰ `java.lang.String`

`String()`	creates an empty string
`int length()`	returns the length of the string
`char charAt(int i)`	returns the character in the string at index `i`
`String substring(int i, int j)`	returns a substring of the string from index `i` (inclusive) to index `j` (exclusive)

Examples of ADTs · String

java.lang.String

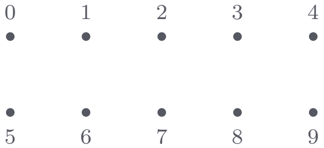
`String()`	creates an empty string
`int length()`	returns the length of the string
`char charAt(int i)`	returns the character in the string at index `i`
`String substring(int i, int j)`	returns a substring of the string from index `i` (inclusive) to index `j` (exclusive)

Example

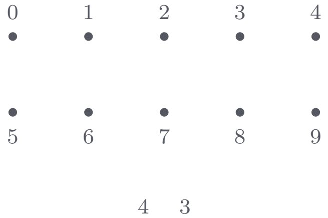
```java
public static boolean isPalindrome(String s) {
    int n = s.length();
    if (n == 0) {
        return true;
    }
    return s.charAt(0) == s.charAt(n - 1) && isPalindrome(s.substring(1, n - 1));
}
```

Examples of ADTs · Union-find

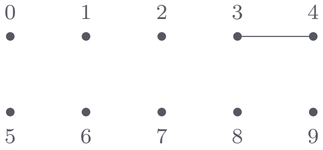
Examples of ADTs · Union-find



Examples of ADTs · Union-find

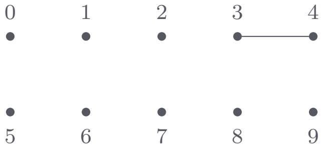


Examples of ADTs · Union-find



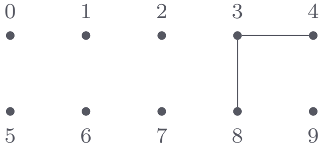
4 3

Examples of ADTs · Union-find



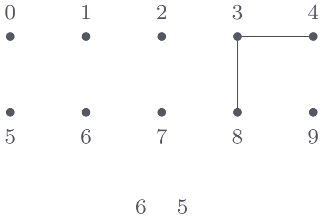
3 8

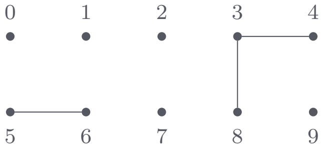
Examples of ADTs · Union-find



3 8

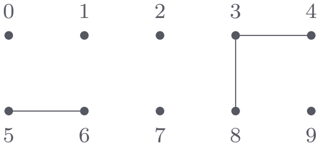
Examples of ADTs · Union-find





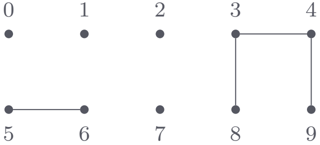
6 5

Examples of ADTs · Union-find



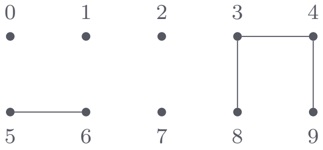
9 4

Examples of ADTs · Union-find



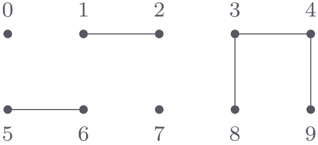
9 4

Examples of ADTs · Union-find

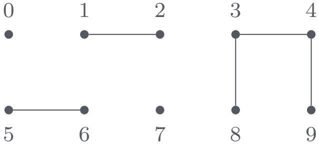


2 1

Examples of ADTs · Union-find

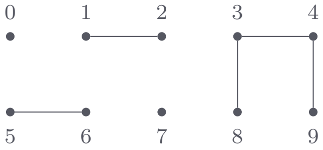


2 1



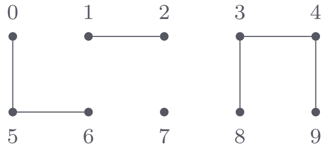
8 9

Examples of ADTs · Union-find



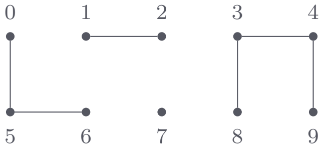
5 0

Examples of ADTs · Union-find



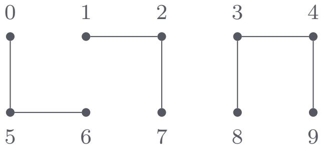
5 0

Examples of ADTs · Union-find



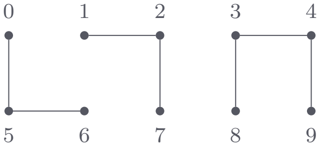
7 2

Examples of ADTs · Union-find



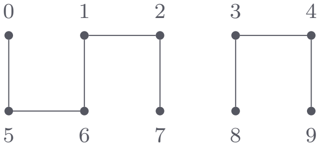
7   2

Examples of ADTs · Union-find



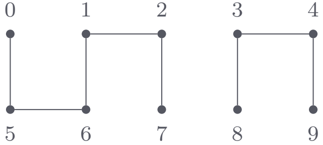
6 1

Examples of ADTs · Union-find



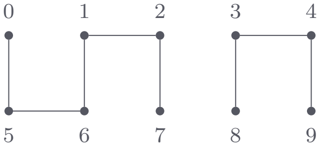
6 1

Examples of ADTs · Union-find



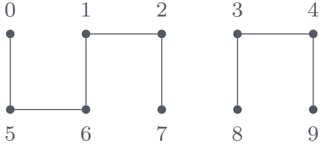
1 0

Examples of ADTs · Union-find



6   7

Examples of ADTs · Union-find



Examples of ADTs · Union-find

Examples of ADTs · Union-find

☰ `WeightedQuickUnionUF implements UF`

`WeightedQuickUnionUF(int n)`	constructs an empty union-find data structure with `n` sites
`int find(int p)`	returns the canonical site of the component containing site `p`
`int count()`	returns the number of components
`boolean connected(int p, int q)`	returns `true` if sites `p` and `q` belong to the same component, and `false` otherwise
`void union(int p, int q)`	connects sites `p` and `q`

Examples of ADTs · Union-find

Examples of ADTs · Union-find

Program: `Components.java`

Examples of ADTs · Union-find

Program: `Components.java`

⇝ Standard input: n (int) and a sequence of pairs of integers representing sites

Examples of ADTs · Union-find

Program: `Components.java`

- ⇝ Standard input: n (int) and a sequence of pairs of integers representing sites
- ⇝ Standard output: number of components left after merging the sites that are in different components

Examples of ADTs · Union-find

Program: `Components.java`

- Standard input: n (int) and a sequence of pairs of integers representing sites
- Standard output: number of components left after merging the sites that are in different components

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

Examples of ADTs · Union-find

Program: `Components.java`

- ⇝ Standard input: n (int) and a sequence of pairs of integers representing sites
- ⇝ Standard output: number of components left after merging the sites that are in different components

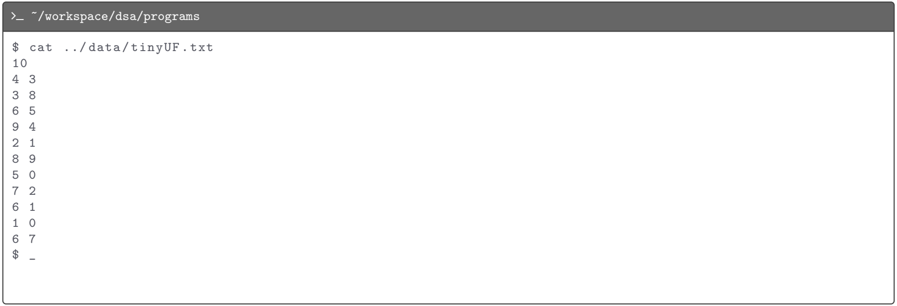
`>_ ~/workspace/dsa/programs`

```
$ cat ../data/tinyUF.txt
```

Examples of ADTs · Union-find

Program: `Components.java`

- ⇝ Standard input: n (int) and a sequence of pairs of integers representing sites
- ⇝ Standard output: number of components left after merging the sites that are in different components



```
>_ ~/workspace/dsa/programs
```

```
$ cat ../data/tinyUF.txt
10
4 3
3 8
6 5
9 4
2 1
8 9
5 0
7 2
6 1
1 0
6 7
$ _
```

Examples of ADTs · Union-find

Program: `Components.java`

- ⇝ Standard input: n (int) and a sequence of pairs of integers representing sites
- ⇝ Standard output: number of components left after merging the sites that are in different components

```
>_ ~/workspace/dsa/programs

$ cat ../data/tinyUF.txt
10
4 3
3 8
6 5
9 4
2 1
8 9
5 0
7 2
6 1
1 0
6 7
$ java Components < ../data/tinyUF.txt
```

Examples of ADTs · Union-find

Program: `Components.java`

- Standard input: n (int) and a sequence of pairs of integers representing sites
- Standard output: number of components left after merging the sites that are in different components

```
~/workspace/dsa/programs

$ cat ../data/tinyUF.txt
10
4 3
3 8
6 5
9 4
2 1
8 9
5 0
7 2
6 1
1 0
6 7
$ java Components < ../data/tinyUF.txt
2 components
$ _
```

Examples of ADTs · Union-find

Examples of ADTs · Union-find

Components.java

```java
import dsa.WeightedQuickUnionUF;
import stdlib.StdIn;
import stdlib.StdOut;

public class Components {
    public static void main(String[] args) {
        int n = StdIn.readInt();
        WeightedQuickUnionUF uf = new WeightedQuickUnionUF(n);
        while (!StdIn.isEmpty()) {
            int p = StdIn.readInt();
            int q = StdIn.readInt();
            uf.union(p, q);
        }
        StdOut.println(uf.count() + " components");
    }
}
```

Defining an ADT

Defining an ADT

Program.java

```java
[package dsa;]

// Import statements.
...

// Class definition.
public class Program [implements <name>] {
    // Field declarations.
    ...

    // Constructor definitions.
    ...

    // Method definitions.
    ...

    // Function definitions.
    ...

    // Inner class definitions.
    ...
}
```

Defining an ADT

Defining an ADT

Field declaration statement

```
private|public [static] <type> <name>;
```

Defining an ADT

Field declaration statement

```
private|public [static] <type> <name>;
```

Fields are accessed as `[<target>].<name>`, where `<target>` is an object name for an instance field and a library name for a `static` field

Defining an ADT

Field declaration statement

```
private|public [static] <type> <name>;
```

Fields are accessed as `[<target>].<name>`, where `<target>` is an object name for an instance field and a library name for a `static` field

Examples:

Defining an ADT

Field declaration statement

```
private|public [static] <type> <name>;
```

Fields are accessed as `[<target>].<name>`, where `<target>` is an object name for an instance field and a library name for a `static` field

Examples:

- Instance fields `String id` and `int count` in `Counter`

Defining an ADT

Field declaration statement

```
private|public [static] <type> <name>;
```

Fields are accessed as `[<target>].<name>`, where `<target>` is an object name for an instance field and a library name for a `static` field

Examples:

- Instance fields `String id` and `int count` in `Counter`
- Static field `double PI` in `Math`

Defining an ADT

Defining an ADT

Constructor definition

```
private|public <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

where `<name>` is the name of the ADT

Defining an ADT

Constructor definition

```
private|public <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

where `<name>` is the name of the ADT

Example

Counter.java

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

Defining an ADT

Constructor definition

```
private|public <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

where `<name>` is the name of the ADT

Example

Counter.java

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

Within a constructor, `this` is a reference to the object being constructed

Defining an ADT

Constructor definition

```
private|public <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

where `<name>` is the name of the ADT

Example

Counter.java

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

Within a constructor, `this` is a reference to the object being constructed

If an ADT has no explicit constructors, `javac` implicitly provides an empty constructor

Defining an ADT

Defining an ADT

Method definition

```
private|public <type> <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

Defining an ADT

Method definition

```
private|public <type> <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

Example

Counter.java

```java
...
    public void increment() {
        count++;
    }

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

Defining an ADT

Method definition

```
private|public <type> <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

Example

Counter.java

```java
...
    public void increment() {
        count++;
    }

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

Within a method, `this` is a reference to the object on which the method was invoked

Defining an ADT · Interface

Defining an ADT · Interface

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API

Defining an ADT · Interface

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API

Example

Defining an ADT · Interface

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API

Example

```java
public interface Animal {
    public String sound();
}
```

Defining an ADT · Interface

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API

Example

```
public interface Animal {
    public String sound();
}
```

```
public class Elephant implements Animal {
    public String sound() {
        return "trumpet";
    }
}
```

Defining an ADT · Interface

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API

Example

```java
public interface Animal {
    public String sound();
}
```

```java
public class Elephant implements Animal {
    public String sound() {
        return "trumpet";
    }
}
```

```java
public class Tiger implements Animal {
    public String sound() {
        return "roar";
    }
}
```

Defining an ADT · Interface

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API

Example

```java
public interface Animal {
    public String sound();
}
```

```java
public class Elephant implements Animal {
    public String sound() {
        return "trumpet";
    }
}
```

```java
public class Tiger implements Animal {
    public String sound() {
        return "roar";
    }
}
```

```java
Animal elephant = new Elephant();
Animal tiger = new Tiger();
StdOut.println("An elephant’s " + elephant.sound() + "!");
StdOut.println("A tiger’s " + tiger.sound() + "!");
```

Defining an ADT · Interface

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API

Example

```java
public interface Animal {
    public String sound();
}
```

```java
public class Elephant implements Animal {
    public String sound() {
        return "trumpet";
    }
}
```

```java
public class Tiger implements Animal {
    public String sound() {
        return "roar";
    }
}
```

```java
Animal elephant = new Elephant();
Animal tiger = new Tiger();
StdOut.println("An elephant's " + elephant.sound() + "!");
StdOut.println("A tiger's " + tiger.sound() + "!");
```

```
An elephant's trumpet!
A tiger's roar!
```

Defining an ADT · Comparison Interfaces

Defining an ADT · Comparison Interfaces

java.lang.Comparable

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

Defining an ADT · Comparison Interfaces

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`

Defining an ADT · Comparable ADT

Defining an ADT · Comparable ADT

ComparableADT.java

```java
import java.util.Comparator;

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

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

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

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

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

Defining an ADT · Comparable ADT

Defining an ADT · Comparable ADT

`Counter implements Comparable<Counter>`

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

Defining an ADT · Comparable ADT

Defining an ADT · Comparable ADT

Program: `Counter.java`

Defining an ADT · Comparable ADT

Program: `Counter.java`

⇝ Command-line input: n (int), $trials$ (int)

Defining an ADT · Comparable ADT

Program: `Counter.java`

$\leadsto$ Command-line input: n (int), $trials$ (int)

$\leadsto$ Standard output: frequencies obtained from $trials$ random draws of numbers from the interval $[0, n)$

Defining an ADT · Comparable ADT

Program: `Counter.java`

- ⇝ Command-line input: n (int), $trials$ (int)
- ⇝ Standard output: frequencies obtained from $trials$ random draws of numbers from the interval $[0, n)$

```
>_ ~/workspace/dsa/programs

$ _
```

Defining an ADT · Comparable ADT

Program: `Counter.java`

- ⇝ Command-line input: n (int), *trials* (int)
- ⇝ Standard output: frequencies obtained from *trials* random draws of numbers from the interval $[0, n)$

```
>_ ~/workspace/dsa/programs

$ java dsa.Counter 2 1000
```

Defining an ADT · Comparable ADT

Program: `Counter.java`

- ⇝ Command-line input: n (int), *trials* (int)
- ⇝ Standard output: frequencies obtained from *trials* random draws of numbers from the interval $[0, n)$

```
>_ ~/workspace/dsa/programs

$ java dsa.Counter 2 1000
501 counter 0
499 counter 1
$ _
```

Defining an ADT · Comparable ADT

Defining an ADT · Comparable ADT

Counter.java

```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;
        count = 0;
    }

    public void increment() {
        count++;
    }

    public int tally() {
        return count;
    }

    public void reset() {
        count = 0;
    }

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

Defining an ADT · Comparable ADT

Counter.java

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

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

    public int compareTo(Counter other) {
        return 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]);
        }
    }
}
```

Defining an ADT · Comparable ADT

Defining an ADT · Comparable ADT

☰ `Transaction implements 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
`static Comparator<Transaction> amountOrder()`	returns a comparator for comparing two transactions by amount

Defining an ADT · Comparable ADT

Defining an ADT · Comparable ADT

Program: `Transaction.java`

Defining an ADT · Comparable ADT

Program: `Transaction.java`

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

Defining an ADT · Comparable ADT

Program: `Transaction.java`

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

```
>_ ~/workspace/dsa/programs
```

```
$ _
```

Defining an ADT · Comparable ADT

Program: `Transaction.java`

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

```
>_ ~/workspace/dsa/programs
```

```
$ java dsa.Transaction
```

Defining an ADT · Comparable ADT

Program: `Transaction.java`

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

```
>_ ~/workspace/dsa/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
$ _
```

Defining an ADT · Comparable ADT

Defining an ADT · Comparable ADT

Transaction.java

```java
package dsa;

import java.util.Arrays;
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+");
        name = a[0];
        date = new Date(a[1]);
        amount = Double.parseDouble(a[2]);
    }

    public String name() {
        return name;
    }

    public Date date() {
        return date;
    }

    public double amount() {
        return amount;
```

Defining an ADT · Comparable ADT

Transaction.java

```java
    }

    public int hashCode() {
        int hash = 1;
        hash = 31 * hash + name.hashCode();
        hash = 31 * hash + date.hashCode();
        hash = 31 * hash + ((Double) amount).hashCode();
        return hash;
    }

    public String toString() {
        return String.format("%-10s %10s %8.2f", name, date, amount);
    }

    public int compareTo(Transaction other) {
        return Double.compare(this.amount, other.amount);
    }

    public static Comparator<Transaction> nameOrder() {
        return new NameOrder();
    }

    public static Comparator<Transaction> dateOrder() {
        return new DateOrder();
    }

    public static Comparator<Transaction> amountOrder() {
        return new AmountOrder();
    }

    private static class NameOrder implements Comparator<Transaction> {
        public int compare(Transaction v, Transaction w) {
            return v.name.compareTo(w.name);
        }
    }
```

Defining an ADT · Comparable ADT

Transaction.java

```java

    private static class DateOrder implements Comparator<Transaction> {
        public int compare(Transaction v, Transaction w) {
            return v.date.compareTo(w.date);
        }
    }

    private static class AmountOrder implements Comparator<Transaction> {
        public int compare(Transaction v, Transaction w) {
            return Double.compare(v.amount, w.amount);
        }
    }

    public static void main(String[] args) {
        Transaction[] transactions = new Transaction[4];
        transactions[0] = new Transaction("Turing   6/17/1990  644.08");
        transactions[1] = new Transaction("Tarjan   3/26/2002 4121.85");
        transactions[2] = new Transaction("Knuth    6/14/1999  288.34");
        transactions[3] = new Transaction("Dijkstra 8/22/2007 2678.40");
        StdOut.println("Unsorted:");
        for (Transaction transaction : transactions) {
            StdOut.println(transaction);
        }
        StdOut.println();
        StdOut.println("Sorted by name:");
        Arrays.sort(transactions, new NameOrder());
        for (Transaction transaction : transactions) {
            StdOut.println(transaction);
        }
        StdOut.println();
        StdOut.println("Sorted by date:");
        Arrays.sort(transactions, new DateOrder());
        for (Transaction transaction : transactions) {
            StdOut.println(transaction);
        }
```

Defining an ADT · Comparable ADT

Transaction.java

```java
        StdOut.println();
        StdOut.println("Sorted by amount:");
        Arrays.sort(transactions, new AmountOrder());
        for (Transaction transaction : transactions) {
            StdOut.println(transaction);
        }
    }
}
```

Defining an ADT · Iteration Interfaces

Defining an ADT · Iteration Interfaces

java.lang.Iterable

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

Defining an ADT · Iteration Interfaces

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
`void remove()`	not supported

Defining an ADT · Iterable ADT

Defining an ADT · Iterable ADT

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

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

Defining an ADT · Iterable ADT

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

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

which is equivalent to

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

Defining an ADT · Iterable ADT

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

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

which is equivalent to

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

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

Defining an ADT · Iterable ADT

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

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

which is equivalent to

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

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

Example

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

Defining an ADT · Iterable ADT

Defining an ADT · Iterable ADT

IterableADT.java

```java
import java.util.Iterator;

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

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

        public Type next() {
            ...
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
    ...
}
```

Defining an ADT · Iterable ADT

Defining an ADT · Iterable ADT

Program: `Words.java`

Defining an ADT · Iterable ADT

Program: `Words.java`

$\rightsquigarrow$ Command-line input: *sentence* (String)

Defining an ADT · Iterable ADT

Program: `Words.java`

⇝ Command-line input: *sentence* (String)

⇝ Standard output: the words in *sentence*, one per line

Defining an ADT · Iterable ADT

Program: `Words.java`

- ⇝ Command-line input: *sentence* (String)
- ⇝ Standard output: the words in *sentence*, one per line

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

Defining an ADT · Iterable ADT

Program: `Words.java`

- ⇝ Command-line input: *sentence* (String)
- ⇝ Standard output: the words in *sentence*, one per line

```
>_ ~/workspace/dsa/programs

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

Defining an ADT · Iterable ADT

Program: `Words.java`

- ⇝ Command-line input: *sentence* (String)
- ⇝ Standard output: the words in *sentence*, one per line

```
>_ ~/workspace/dsa/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
$ _
```

Defining an ADT · Iterable ADT

Defining an ADT · Iterable ADT

Words.java

```java
import java.util.Iterator;

import stdlib.StdOut;

public class Words implements Iterable<String> {
    private String sentence;

    public Words(String sentence) {
        this.sentence = sentence;
    }

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

    private class WordsIterator implements Iterator<String> {
        private String[] words;
        private int i;

        public WordsIterator() {
            words = sentence.split("\s+");
            i = 0;
        }

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

        public String next() {
            return words[i++];
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
```

Defining an ADT · Iterable ADT

Words.java

```java
    }

    public static void main(String[] args) {
        String sentence = args[0];
        Words words = new Words(sentence);
        for (String word : words) {
            StdOut.println(word);
        }
    }
}
```

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

Examples: `ArrayIndexOutOfBoundsException` and `NullPointerException`

Error Handling

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

Examples: `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

Examples: `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>
    ...
}
<statement>
...
```

Error Handling

Error Handling

Program: `ErrorHandling.java`

Error Handling

Program: `ErrorHandling.java`

$\rightsquigarrow$ Command-line input: x (double)

Error Handling

Program: `ErrorHandling.java`

- ⇝ Command-line input: x (double)
- ⇝ Standard output: the square root of x

Error Handling

Program: `ErrorHandling.java`

- ⇝ Command-line input: x (double)
- ⇝ Standard output: the square root of x

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

Error Handling

Program: `ErrorHandling.java`

- ⇝ Command-line input: x (double)
- ⇝ Standard output: the square root of x

```
>_ ~/workspace/dsa/programs
```

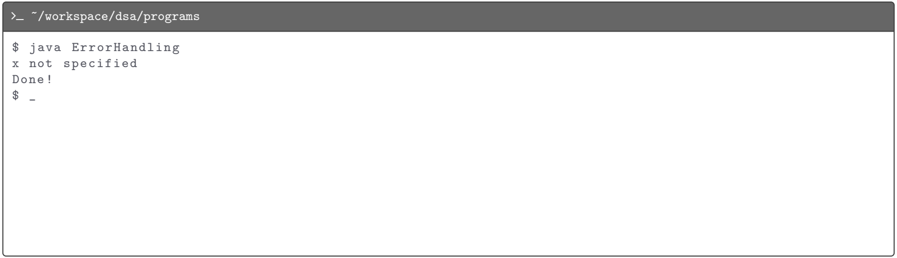
```
$ java ErrorHandling
```

Error Handling

Program: `ErrorHandling.java`

⇝ Command-line input: x (double)

⇝ Standard output: the square root of x



```
>_ ~/workspace/dsa/programs
```

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

Error Handling

Program: `ErrorHandling.java`

- Command-line input: x (double)
- Standard output: the square root of x

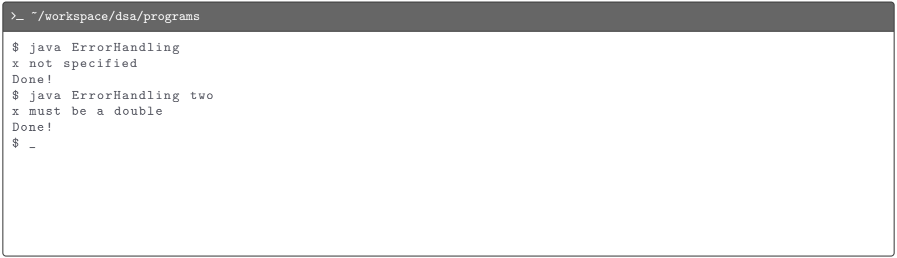
```
>_ ~/workspace/dsa/programs

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

Error Handling

Program: `ErrorHandling.java`

- ⇝ Command-line input: x (double)
- ⇝ Standard output: the square root of x



```
>_ ~/workspace/dsa/programs
```

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

Error Handling

Program: `ErrorHandling.java`

- ⇝ Command-line input: x (double)
- ⇝ Standard output: the square root of x

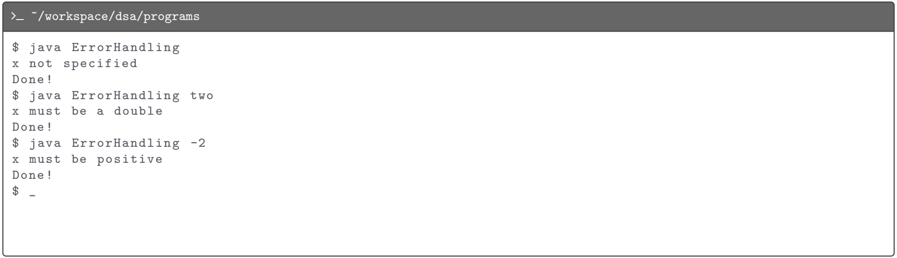
```
>_ ~/workspace/dsa/programs
```

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

Error Handling

Program: `ErrorHandling.java`

- Command-line input: x (double)
- Standard output: the square root of x



```
$ 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

Program: `ErrorHandling.java`

- ⇝ Command-line input: x (double)
- ⇝ Standard output: the square root of x

```
>_ ~/workspace/dsa/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

Program: `ErrorHandling.java`

⇝ Command-line input: x (double)

⇝ Standard output: the square root of x

~/workspace/dsa/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

Error Handling

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);
    }
}
```