Data Structures and Algorithms in Java

Object-oriented Programming: Using Data Types

Outline

Classes

Classes

A class (aka, reference data type) is a blueprint for creating objects

Classes

A class (aka, reference data type) is a blueprint for creating objects

A class defines the properties (attributes) and behaviors (methods) that objects of that class will have

Classes

A class (aka, reference data type) is a blueprint for creating objects

A class defines the properties (attributes) and behaviors (methods) that objects of that class will have

Objects of a class are created by calling constructors

Classes

- A class (aka, reference data type) is a blueprint for creating objects
- A class defines the properties (attributes) and behaviors (methods) that objects of that class will have
- Objects of a class are created by calling constructors
- Constructors are methods without a return type (not even `void`) and with the same name as the class

Classes

- A class (aka, reference data type) is a blueprint for creating objects
- A class defines the properties (attributes) and behaviors (methods) that objects of that class will have
- Objects of a class are created by calling constructors
- Constructors are methods without a return type (not even `void`) and with the same name as the class
- Methods are called on objects

Classes

- A class (aka, reference data type) is a blueprint for creating objects
- A class defines the properties (attributes) and behaviors (methods) that objects of that class will have
- Objects of a class are created by calling constructors
- Constructors are methods without a return type (not even `void`) and with the same name as the class
- Methods are called on objects
- The API for a class describes its behavior

The `Object` class is the root of the class hierarchy

java.lang.Object	
`Object()`	constructs an object
`boolean equals(Object other)`	returns `true` if the object and `other` have the same memory address, and `false` otherwise
`int hashCode()`	returns the memory address of the object
`String toString()`	returns the memory address of the object as a string

The `Object` class is the root of the class hierarchy

java.lang.Object	
`Object()`	constructs an object
`boolean equals(Object other)`	returns `true` if the object and `other` have the same memory address, and `false` otherwise
`int hashCode()`	returns the memory address of the object
`String toString()`	returns the memory address of the object as a string

Every class either directly or indirectly inherits from the `Object` class

The `Object` class is the root of the class hierarchy

java.lang.Object	
`Object()`	constructs an object
`boolean equals(Object other)`	returns `true` if the object and `other` have the same memory address, and `false` otherwise
`int hashCode()`	returns the memory address of the object
`String toString()`	returns the memory address of the object as a string

Every class either directly or indirectly inherits from the `Object` class

A class can override the `Object` class methods to provide more specific behavior

`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

Classes · Example (`String` Class)

java.lang.String	
`String()`	constructs an empty string
`char charAt(int i)`	returns the character in the string at index `i`
`boolean endsWith(String suffix)`	returns `true` if the string ends with the string `suffix`, and `false` otherwise
`boolean equals(Object other)`	returns `true` if the string is the same as `other`, and `false` otherwise
`int length()`	returns the length of the string
`boolean startsWith(String prefix)`	returns `true` if the string starts with the string `prefix`, and `false` otherwise
`String substring(int i, int j)`	returns a substring of the string from index `i` (inclusive) to index `j` (exclusive)

Objects

Objects

An object of a class is created by calling a constructor of that class

```
<type> <name> = new <type>(<arg1>, <arg2>, ...);
```

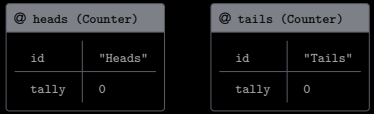
Objects

An object of a class is created by calling a constructor of that class

```
<type> <name> = new <type>(<arg1>, <arg2>, ...);
```

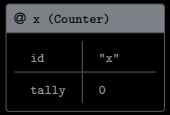
Example

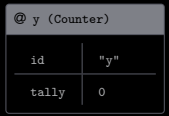
```
Counter heads = new Counter("Heads");
Counter tails = new Counter("Tails");
```



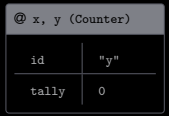
```
Counter x = new Counter("x");
Counter y = new Counter("y");
```

```
Counter x = new Counter("x");
Counter y = new Counter("y");
```






```
x = y;
```



Methods

Methods

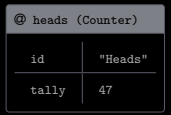
A method is called on an object as `<object>.<name>(<arg1>, <arg2>, ...)`

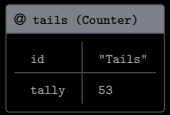
Methods

A method is called on an object as `<object>.<name>(<arg1>, <arg2>, ...)`

Example

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

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());
```





writes

```
47
53
```

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

Two objects `x` and `y` can be tested 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));
```

writes

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

If `s` is a string and `o` is an arbitrary object, `s + o` is equivalent to `s + o.toString()`

- If `s` is a string and `o` is an arbitrary object, `s + o` is equivalent to `s + o.toString()`
- If `o` is an arbirary object, `StdOut.println(o)` is equivalent to `StdOut.println(o.toString())`

`Flips.java`

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

Flips.java

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

>_ ~/workspace/dsaj/programs

```
$ _
```

Examples · Coin Flips I

Flips.java

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

~/workspace/dsaj/programs

```
$ java Flips 1000000
```

Examples · Coin Flips I

Flips.java

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

>_ ~/workspace/dsaj/programs

```
$ java Flips 1000000
500068 Heads
499932 Tails
delta: 136
$ _
```

Examples · Coin Flips I

</> 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()));
    }
}
```

FlipsMax.java

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

FlipsMax.java

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

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

$ _
```

FlipsMax.java

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

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

$ java FlipsMax 1000000
```

FlipsMax.java

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

~/workspace/dsaj/programs

```
$ java FlipsMax 1000000
501214 Tails wins
$ _
```

FlipsMax.java

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

~/workspace/dsaj/programs

```
$ java FlipsMax 1000000
501214 Tails wins
$ java FlipsMax 1000000
```

FlipsMax.java

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

~/workspace/dsaj/programs

```
$ java FlipsMax 1000000
501214 Tails wins
$ java FlipsMax 1000000
500222 Heads wins
$ _
```

FlipsMax.java

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

~/workspace/dsaj/programs

```
$ java FlipsMax 1000000
501214 Tails wins
$ java FlipsMax 1000000
500222 Heads wins
$ java FlipsMax 1000000
```

FlipsMax.java

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

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

$ java FlipsMax 1000000
501214 Tails wins
$ java FlipsMax 1000000
500222 Heads wins
$ java FlipsMax 1000000
500846 Tails wins
$ _
```

Examples · Coin Flips II

</> FlipsMax.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) {
        return x.tally() > y.tally() ? x : y;
    }
}
```

Rolls.java

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

Rolls.java

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

>_ ~/workspace/dsaj/programs

```
$ _
```

Rolls.java

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

>_ ~/workspace/dsaj/programs

```
$ java Rolls 1000000
```

Rolls.java

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

>_ ~/workspace/dsaj/programs

```
$ java Rolls 1000000
166383 1s
166911 2s
166561 3s
166630 4s
166993 5s
166522 6s
$ _
```

Examples · Die Rolls

</> 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]);
        }
    }
}
```

PotentialGene.java

Command-line input	*dna* (String)
Standard output	whether *dna* corresponds to a potential gene or not

PotentialGene.java

Command-line input	*dna* (String)
Standard output	whether *dna* corresponds to a potential gene or not

>_ ~/workspace/dsaj/programs

```
$ _
```

PotentialGene.java

Command-line input	*dna* (String)
Standard output	whether *dna* corresponds to a potential gene or not

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

$ java PotentialGene ATGCGCCTGCGTCTGTACTAG
```

PotentialGene.java

Command-line input	*dna* (String)
Standard output	whether *dna* corresponds to a potential gene or not

~/workspace/dsaj/programs

```
$ java PotentialGene ATGCGCCTGCGTCTGTACTAG
true
$ _
```

PotentialGene.java

Command-line input	*dna* (String)
Standard output	whether *dna* corresponds to a potential gene or not

~/workspace/dsaj/programs

```
$ java PotentialGene ATGCGCCTGCGTCTGTACTAG
true
$ java PotentialGene ATGCGCTGCGTCTGTACTAG
```

PotentialGene.java

Command-line input	*dna* (String)
Standard output	whether *dna* corresponds to a potential gene or not

~/workspace/dsaj/programs

```
$ java PotentialGene ATGCGCCTGCGTCTGTACTAG
true
$ java PotentialGene ATGCGCTGCGTCTGTACTAG
false
$ _
```

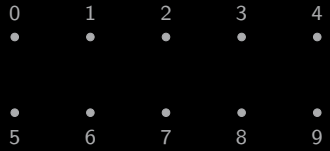
Examples · Potential Gene

</> PotentialGene.java

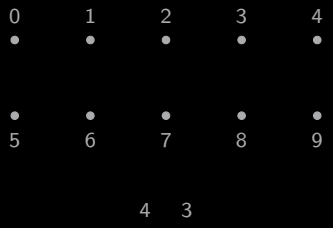
```java
import stdlib.StdOut;

public class PotentialGene {
    public static void main(String[] args) {
        String dna = args[0];
        StdOut.println(isPotentialGene(dna));
    }

    private static boolean isPotentialGene(String dna) {
        String ATG = "ATG", TAA = "TAA", TAG = "TAG", TGA = "TGA";
        if (dna.length() % 3 != 0) {
            return false;
        }
        if (!dna.startsWith(ATG)) {
            return false;
        }
        for (int i = 3; i < dna.length() - 3; i++) {
            if (i % 3 == 0) {
                String codon = dna.substring(i, i + 3);
                if (codon.equals(TAA) || codon.equals(TAG) || codon.equals(TGA)) {
                    return false;
                }
            }
        }
        return dna.endsWith(TAA) || dna.endsWith(TAG) || dna.endsWith(TGA);
    }
}
```

Dynamic Connectivity Problem

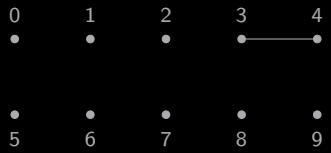
Dynamic Connectivity Problem



Dynamic Connectivity Problem

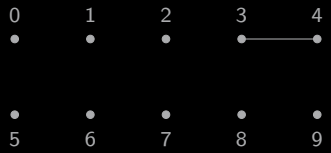


Dynamic Connectivity Problem



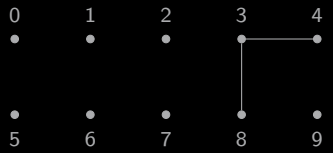
4 3

Dynamic Connectivity Problem



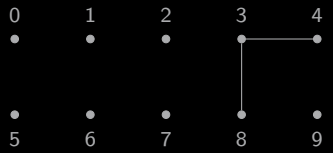
3 8

Dynamic Connectivity Problem



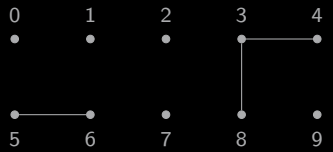
3 8

Dynamic Connectivity Problem



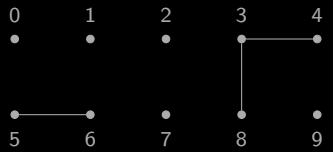
6 5

Dynamic Connectivity Problem



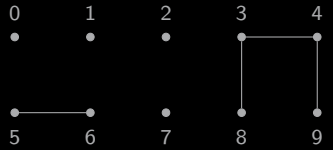
6 5

Dynamic Connectivity Problem



9 4

Dynamic Connectivity Problem

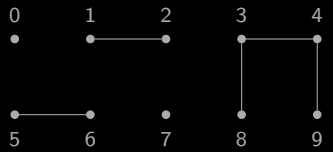


9 4

Dynamic Connectivity Problem

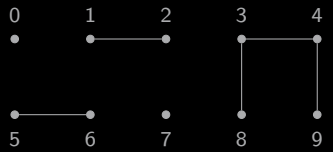
2 1

Dynamic Connectivity Problem



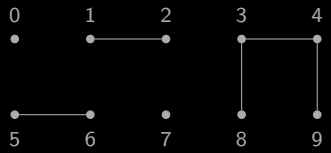
2 1

Dynamic Connectivity Problem



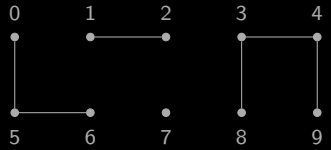
8 9

Dynamic Connectivity Problem



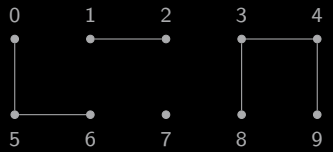
5 0

Dynamic Connectivity Problem



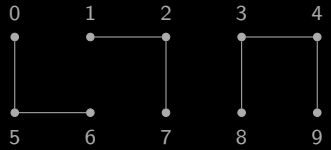
5 0

Dynamic Connectivity Problem



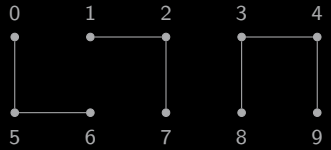
7 2

Dynamic Connectivity Problem



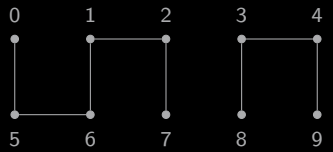
7 2

Dynamic Connectivity Problem



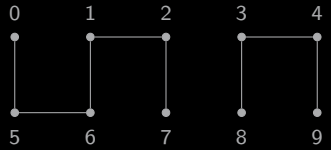
6 1

Dynamic Connectivity Problem



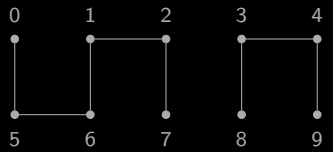
6 1

Dynamic Connectivity Problem



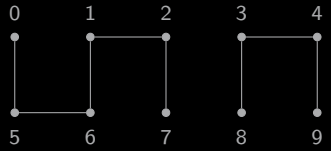
1 0

Dynamic Connectivity Problem



6 7

Dynamic Connectivity Problem



Dynamic Connectivity Problem

Dynamic Connectivity Problem

Notation

- Number of sites, n
- Site identifier, $i \in [0, n)$
- Component identifier, $i \in [0, n)$

Dynamic Connectivity Problem

Dynamic Connectivity Problem

dsa.WeightedQuickUnionUF implements dsa.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`

Dynamic Connectivity Problem

Dynamic Connectivity Problem

DynamicConnectivity.java

Standard input	n (int) and a sequence of pairs of integers
Standard output	the pairs, whether they were merged, and if so, the number of components left

Dynamic Connectivity Problem

DynamicConnectivity.java

Standard input	n (int) and a sequence of pairs of integers
Standard output	the pairs, whether they were merged, and if so, the number of components left

>_ ~/workspace/dsaj

```
$ _
```

Dynamic Connectivity Problem

DynamicConnectivity.java

Standard input	n (int) and a sequence of pairs of integers
Standard output	the pairs, whether they were merged, and if so, the number of components left

>_ ~/workspace/dsaj

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

Dynamic Connectivity Problem

DynamicConnectivity.java

Standard input	*n* (int) and a sequence of pairs of integers
Standard output	the pairs, whether they were merged, and if so, the number of components left

~/workspace/dsaj

```
$ cat data/tinyUF.txt
10
4 3
3 8
...
6 7
$ _
```

Dynamic Connectivity Problem

DynamicConnectivity.java

Standard input	*n* (int) and a sequence of pairs of integers
Standard output	the pairs, whether they were merged, and if so, the number of components left

~/workspace/dsaj

```
$ cat data/tinyUF.txt
10
4 3
3 8
...
6 7
$ java DynamicConnectivity < data/tinyUF.txt
```

Dynamic Connectivity Problem

DynamicConnectivity.java

Standard input	n (int) and a sequence of pairs of integers
Standard output	the pairs, whether they were merged, and if so, the number of components left

>_ ~/workspace/dsaj

```
$ cat data/tinyUF.txt
10
4 3
3 8
...
6 7
$ java DynamicConnectivity < data/tinyUF.txt
4 3 [merged, 9 components]
3 8 [merged, 8 components]
6 5 [merged, 7 components]
9 4 [merged, 6 components]
2 1 [merged, 5 components]
8 9
5 0 [merged, 4 components]
7 2 [merged, 3 components]
6 1 [merged, 2 components]
1 0
6 7
$ _
```

Dynamic Connectivity Problem

Dynamic Connectivity Problem

</> DynamicConnectivity.java

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

public class DynamicConnectivity {
    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();
            StdOut.printf("%d %d", p, q);
            if (uf.connected(p, q)) {
                StdOut.println();
                continue;
            }
            uf.union(p, q);
            StdOut.println(" [merged, " + uf.count() + " components]");
        }
    }
}
```

File Input and Output

File Input and Output

stdlib.In

`In(String name)`	constructs an input stream from a file with the given name
`boolean hasNextLine()`	returns `true` if the input stream has a next line, and `false` otherwise
`String readLine()`	reads and returns the next line from the input stream
`void close()`	closes the input stream

File Input and Output

File Input and Output

stdlib.Out

`Out(String name)`	constructs an output stream from a file with the given name
`void println(Object x)`	prints an object and a newline to the output stream
`void close()`	closes the output stream

File Input and Output

File Input and Output

Copy.java

Command-line input	*from* (String) and *to* (String)
File output	file with name *to* with the contents of the file with name *from*

File Input and Output

Copy.java

Command-line input	*from* (String) and *to* (String)
File output	file with name *to* with the contents of the file with name *from*

>_ ~/workspace/dsaj/programs

```
$ _
```

File Input and Output

Copy.java

Command-line input	*from* (String) and *to* (String)
File output	file with name *to* with the contents of the file with name *from*

>_ ~/workspace/dsaj/programs

```
$ java Copy data/tale.txt ./taleCopy.txt
```

File Input and Output

Copy.java

Command-line input	*from* (String) and *to* (String)
File output	file with name *to* with the contents of the file with name *from*

~/workspace/dsaj/programs

```
$ java Copy data/tale.txt ./taleCopy.txt
$ _
```

File Input and Output

Copy.java

Command-line input	*from* (String) and *to* (String)
File output	file with name *to* with the contents of the file with name *from*

~/workspace/dsaj/programs

```
$ java Copy data/tale.txt ./taleCopy.txt
$ head -5 taleCopy.txt
```

File Input and Output

Copy.java

Command-line input	*from* (String) and *to* (String)
File output	file with name *to* with the contents of the file with name *from*

>_ ~/workspace/dsaj/programs

```
$ java Copy data/tale.txt ./taleCopy.txt
$ head -5 taleCopy.txt
it was the best of times it was the worst of times
it was the age of wisdom it was the age of foolishness
it was the epoch of belief it was the epoch of incredulity
it was the season of light it was the season of darkness
it was the spring of hope it was the winter of despair
$ _
```

File Input and Output

File Input and Output

</> Copy.java

```java
import stdlib.In;
import stdlib.Out;

public class Copy {
    public static void main(String[] args) {
        String from = args[0];
        String to = args[1];
        In in = new In(from);
        Out out = new Out(to);
        while (in.hasNextLine()) {
            String line = in.readLine();
            out.println(line);
        }
        in.close();
        out.close();
    }
}
```