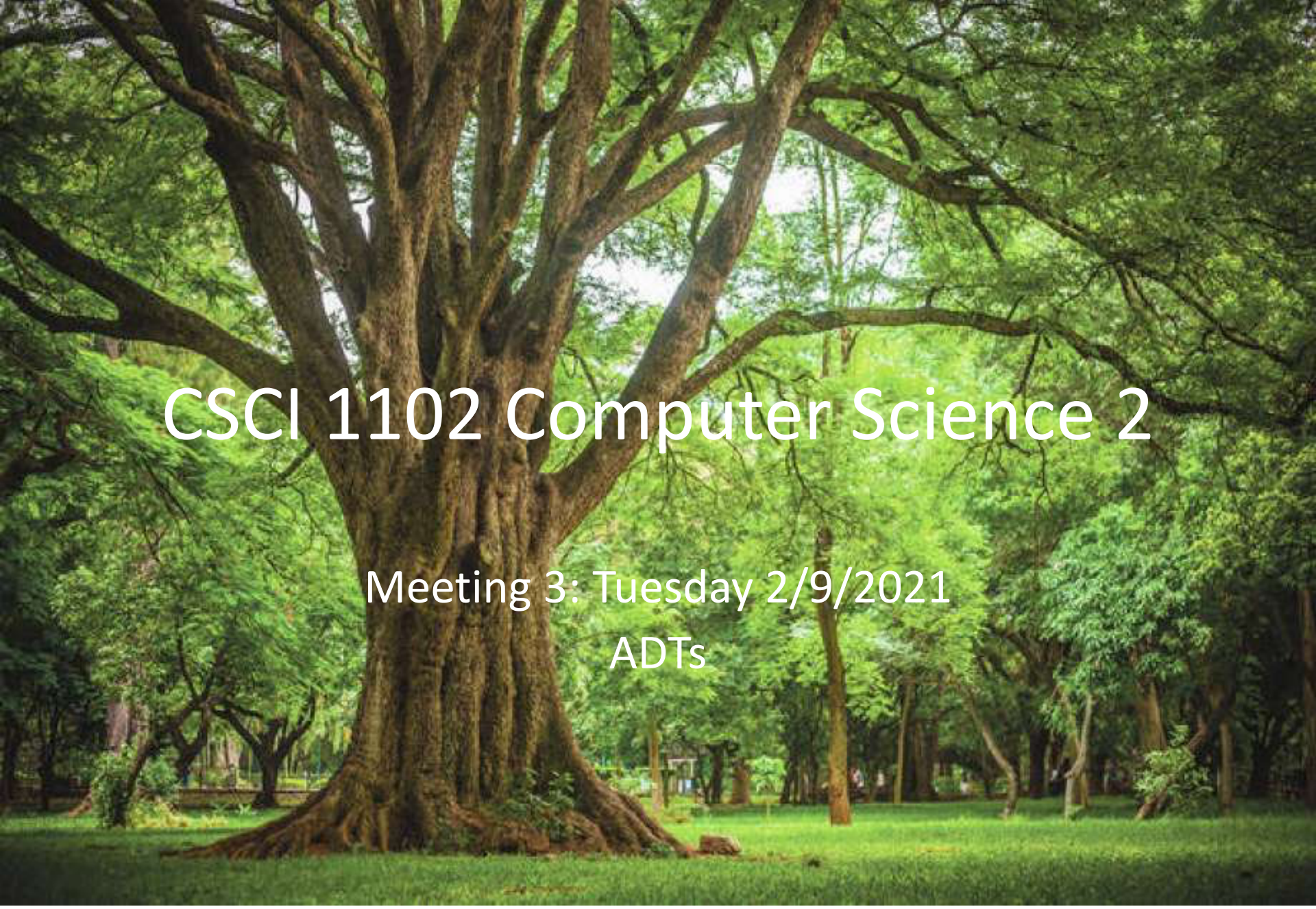


CSCI 1102 Computer Science 2

Meeting 3: Tuesday 2/9/2021

ADTs

Today

- A few stray notes
- Abstract Data Types

Stray Notes

- Repos vs IntelliJ projects
- jshell

Function Call Syntax

static vs dynamic functions

- Static functions are accessed via their containing class symbol and called using standard function call notation.

```
10
11
12      int i = Integer.parseInt("343");
13
14
```

static vs dynamic functions

- Dynamic functions are accessed via a value constructed with the new keyword and are called using message-passing style.

```
3     Integer i = new Integer(343);
4     int result = i.compareTo(12);
5
6     // Or
7
8     int result = new Integer(343).compareTo(12)
```

Scope

```java
public class Code {

  public int i;

  public static void main(String[] args) {

    int j = 1;

    if (args.length == 2) {
      int k = j;
      k = k * 2;
    }
  }
}
```

```
public class Code {

  public int i;

  public static void main(String[] args) {

    int j = 1;

    if (args.length == 2) {
      int k = j;
      k = k * 2;
    }
  }
}
```

i in scope.
args, j & k not in scope.

```
public class Code {

  public int i;

  public static void main(String[] args) {

    int j = 1;

    if (args.length == 2) {
      int k = j;
      k = k * 2;
    }
  }
}
```

i in scope.

i, args & j in scope.
k not in scope.

```
public class Code {

  public int i;

  public static void main(String[] args) {

    int j = 1;

    if (args.length == 2) {
      int k = j;
      k = k * 2;
    }
  }
}
```

i in scope.

i, args & j in scope.

i, args, j & k in scope.

Casting

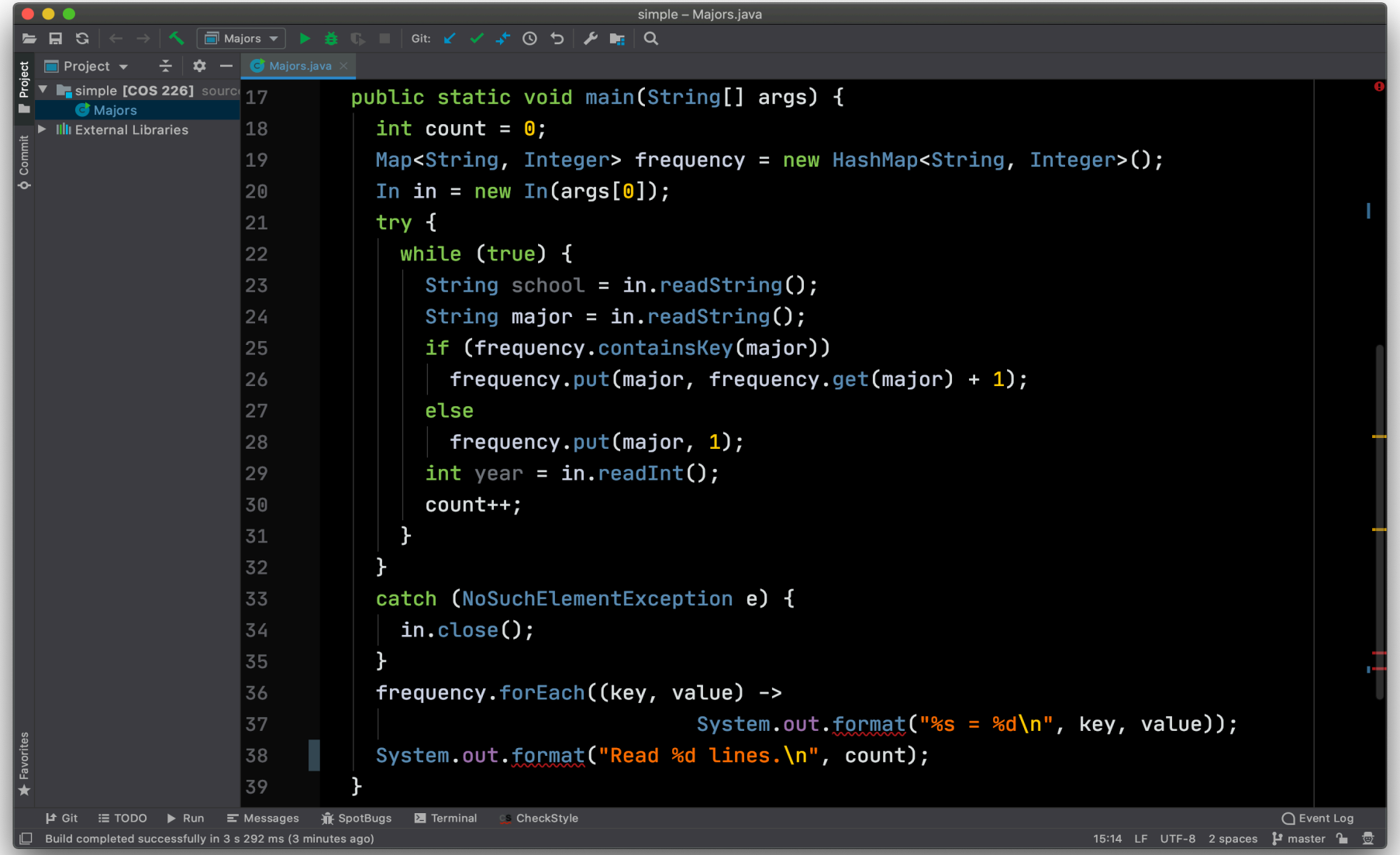
Base Types & Wrapper Classes

Base Type	Wrapper Class/Type
int (32-bit integers)	Integer
long (64-bit integers)	Long
short (16-bit integers)	Short
byte (8-bit integers)	Byte
float (32-bit floats)	Float
double (64-bit floats)	Double
char	Character
boolean	Boolean

Casting

```
15
16    static double average(int m, int n) {
17      return (double) (m + n) / 2.0;
18    }
19
```

Compute a Frequency Table of Majors



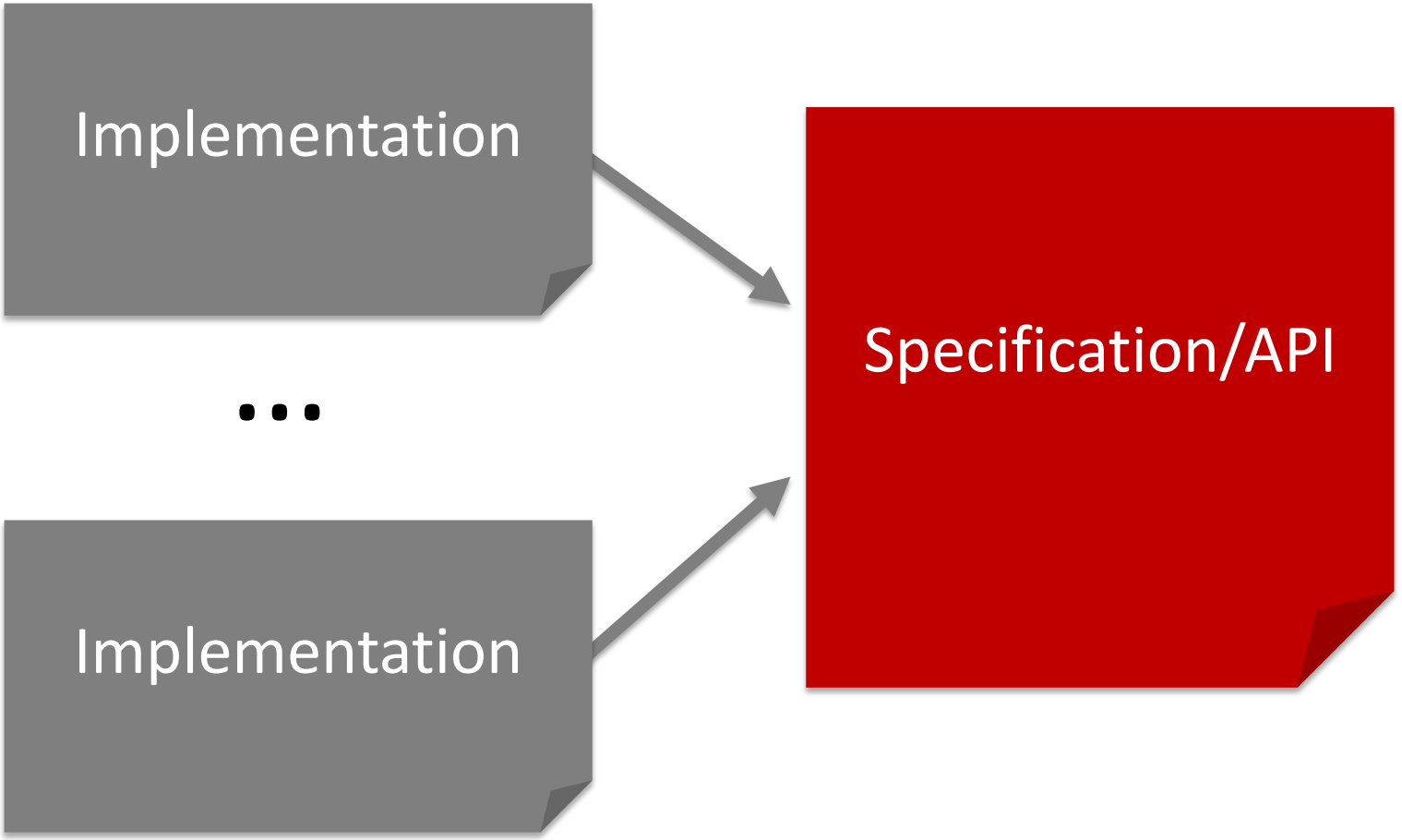
Abstract Data Types

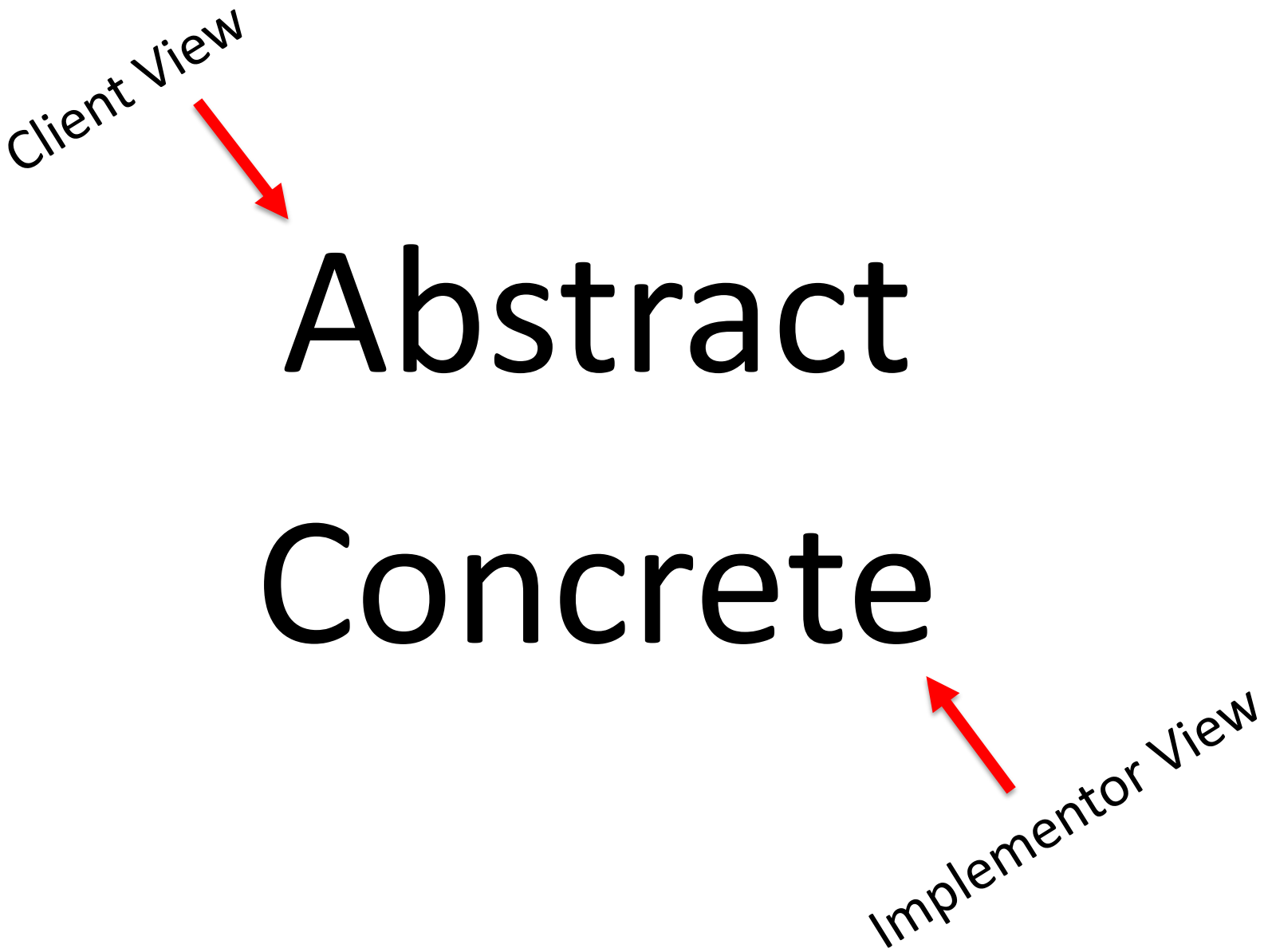
ADTs (B. Liskov)

- We decide we need a new type;
- Specify the name of the type and the type signatures of the operations;
- **Then** develop one or more implementations.

ADTs

ADTs




Client View
Abstract
Concrete
Implementor View

ADTs & Information Hiding

- The client knows only the type signatures of the operations and their performance properties.
- The owner is free to manage the implementation.

ADTs in Java

- Specification: use an interface;
- Implementation: use a class.

A Draft Frequency Type - the API

```
14
15  public interface Frequency {
16
17    int size();
18
19    int get(String key);
20
21    boolean containsKey(String key);
22
23    void put(String key, int value);
24
25    void forEach(java.util.function.BiConsumer<String, Integer> operation);
26
27    Collection<Integer> values();
28
29    Set<String> keys();
30  }
```

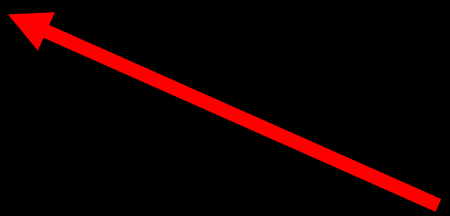
Type signatures for seven dynamic functions.
Implicitly public.

The Frequency Type – an Implementation

```
16  public class FrequencyC implements Frequency {
17
18    ....
19
20    public int get(String key) { ... }
21
22    public boolean contains(String key) { ... }
23
24    public void put(String key, int value) { ... }
25
26    ....
27  }
28
```

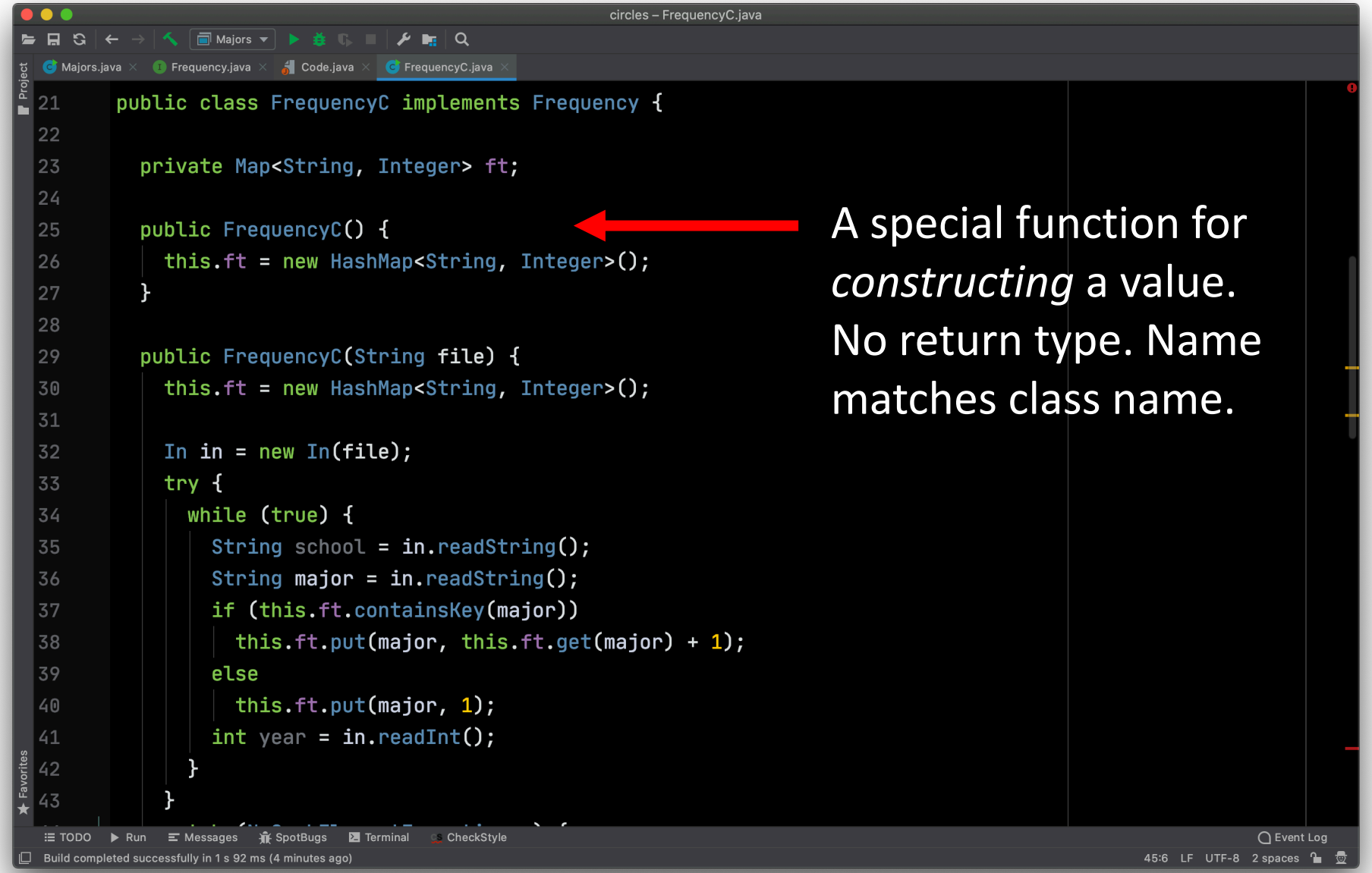
The Frequency Type – an Implementation

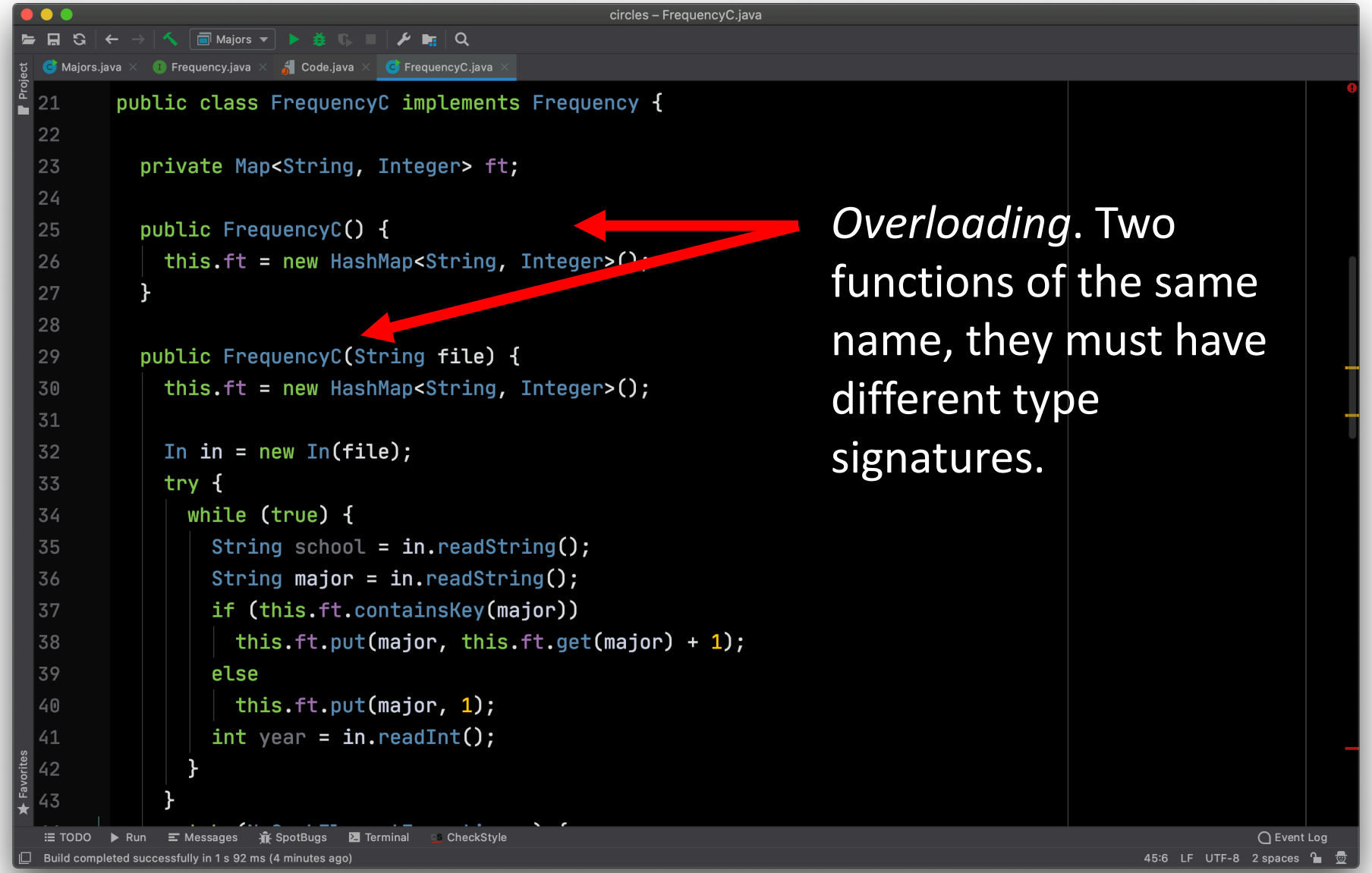
```
16  public class FrequencyC implements Frequency {
17
18    ....
19
20    public int get(String key) { .... }
21
22    public boolean contains(String key) { .... }
23
24    public void put(String key, int value) { .... }
25
26    ....
27  }
28
```



Instructs the Java compiler to confirm that this implementation agrees with the type signature specified in the interface. If it doesn't, the program is rejected.

Constructors

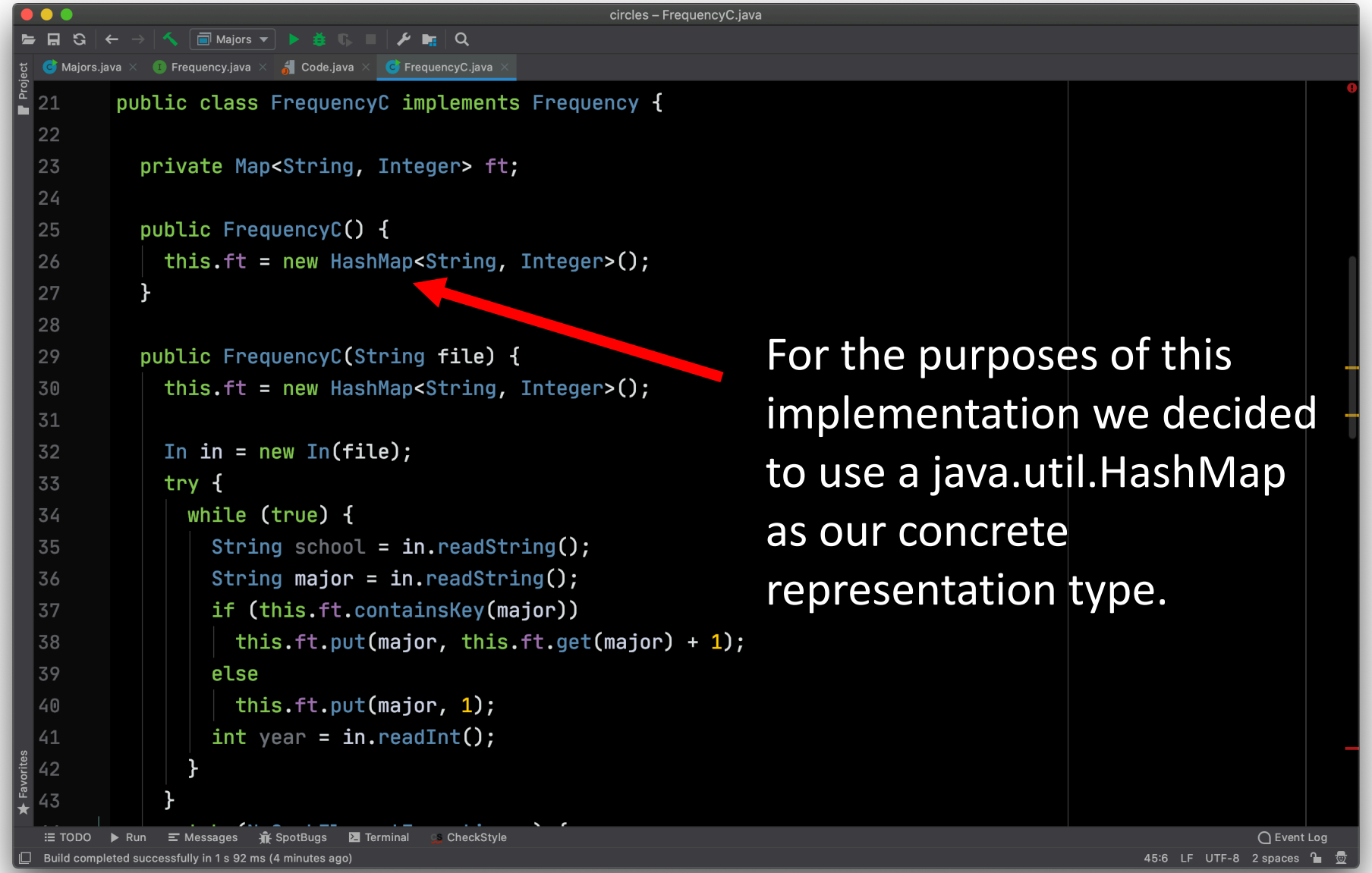

circles – FrequencyC.java
Majors
Majors.java
Frequency.java
Code.java
FrequencyC.java
21 public class FrequencyC implements Frequency {
22
23 private Map<String, Integer> ft;
24
25 public FrequencyC() {
26 this.ft = new HashMap<String, Integer>();
27 }
28
29 public FrequencyC(String file) {
30 this.ft = new HashMap<String, Integer>();
31
32 In in = new In(file);
33 try {
34 while (true) {
35 String school = in.readString();
36 String major = in.readString();
37 if (this.ft.containsKey(major))
38 this.ft.put(major, this.ft.get(major) + 1);
39 else
40 this.ft.put(major, 1);
41 int year = in.readInt();
42 }
43 }
A special function for *constructing* a value. No return type. Name matches class name.
TODO
Run
Messages
SpotBugs
Terminal
CheckStyle
Event Log
Build completed successfully in 1 s 92 ms (4 minutes ago)
45:6 LF UTF-8 2 spaces



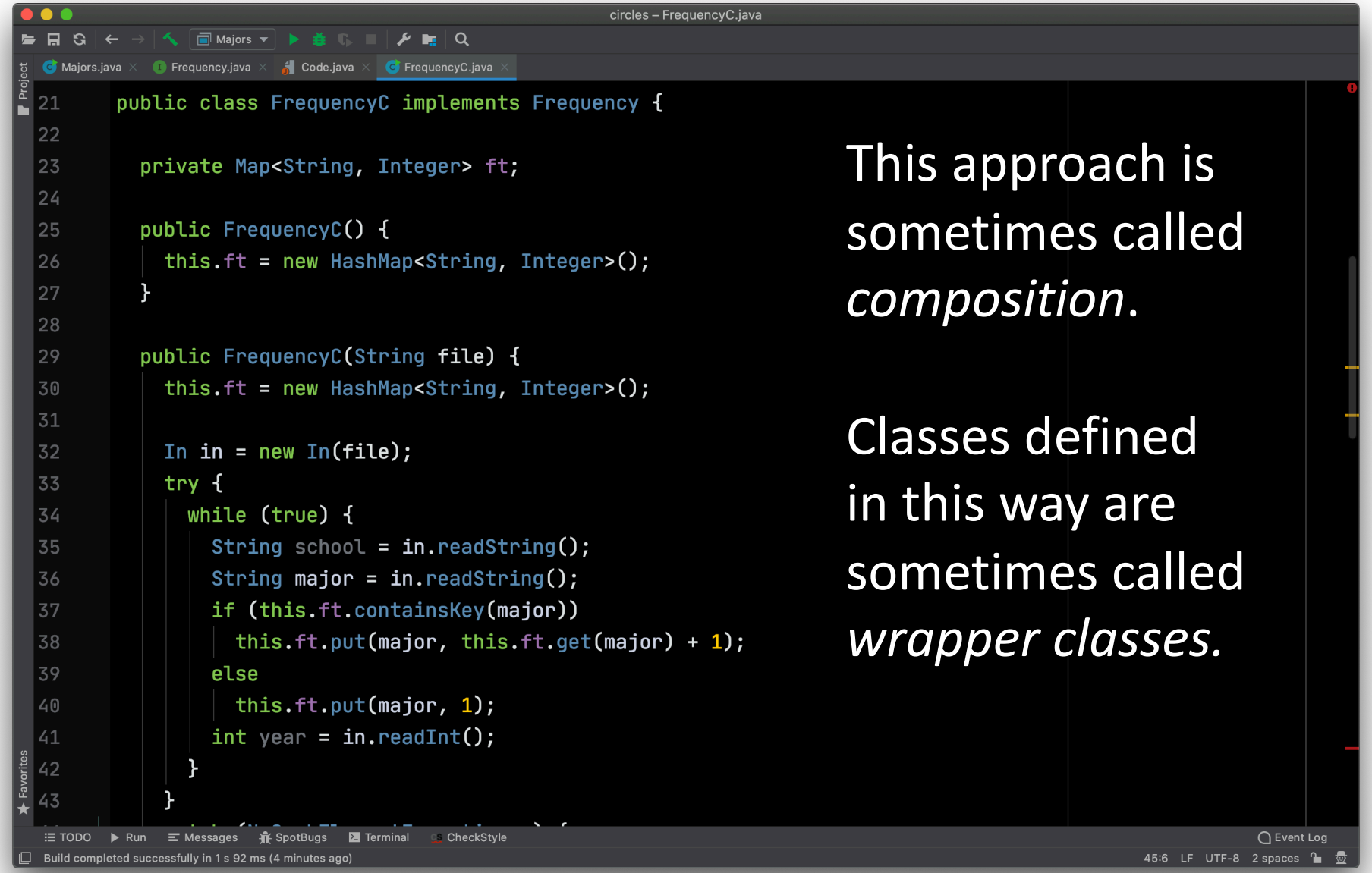
Overloading. Two functions of the same name, they must have different type signatures.

Building New Types Compositionally


circles – FrequencyC.java
Majors
Majors.java
Frequency.java
Code.java
FrequencyC.java
21 public class FrequencyC implements Frequency {
22
23 private Map<String, Integer> ft;
24
25 public FrequencyC() {
26 this.ft = new HashMap<String, Integer>();
27 }
28
29 public FrequencyC(String file) {
30 this.ft = new HashMap<String, Integer>();
31
32 In in = new In(file);
33 try {
34 while (true) {
35 String school = in.readString();
36 String major = in.readString();
37 if (this.ft.containsKey(major))
38 this.ft.put(major, this.ft.get(major) + 1);
39 else
40 this.ft.put(major, 1);
41 int year = in.readInt();
42 }
43 }
A *private* instance variable. The concrete representation type is Map<String, Integer>.
TODO
Run
Messages
SpotBugs
Terminal
CheckStyle
Event Log
Build completed successfully in 1 s 92 ms (4 minutes ago)
45:6 LF UTF-8 2 spaces



For the purposes of this implementation we decided to use a java.util.HashMap as our concrete representation type.



This approach is sometimes called *composition*.

Classes defined in this way are sometimes called *wrapper classes.*

```
44
45      public int get(String key) {
46        return ft.get(key);
47      }
48
49      public boolean containsKey(String key) {
50        return ft.containsKey(key);
51      }
52
53      public void put(String key, int value) {
54        ft.put(key, value);
55      }
56
57      public void forEach(BiConsumer<String, Integer> opn) {
58        ft.forEach(opn);
59      }
60
61      public Collection<Integer> values() {
62        return ft.values();
63      }
```

Forwarding or *delegation*.

Two Main Themes of CS1

- How to design a reasonable API for an ADT
- How to choose concrete representation types in the implementation of the ADT so that the required operations are performant.
- Different applications of the ADT may have different performance requirements.

A Note on λ and Friends

```java
private static void myPrinter(String s, Integer i) {
  System.out.format("%s = %d", s, i);
}

public static void main(String[] args) {

  Frequency frequency = new FrequencyC();

  frequency.put("Alice", 20);
  frequency.put("Bob", 10);

  frequency.forEach((key, value) -> System.out.format("%s = %d", key, value));


}
```

λ an anonymous function, no type information required for the parameters.

```java
private static void myPrinter(String s, Integer i) {
  System.out.format("%s = %d", s, i);
}

public static void main(String[] args) {

  Frequency frequency = new FrequencyC();

  frequency.put("Alice", 20);
  frequency.put("Bob", 10);

  frequency.forEach((key, value) -> System.out.format("%s = %d", key, value));
  frequency.forEach(FrequencyC::myPrinter);

}
```

A static function defined in class FrequencyC

:: notation