Principles of Software Construction: Objects, Design, and Concurrency

Introduction to Java

Charlie Garrod **Chris Timperley**

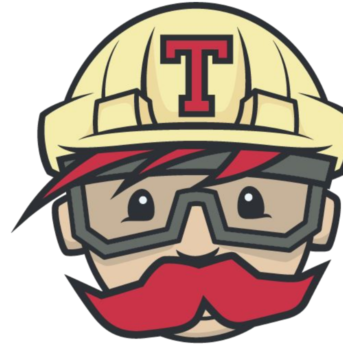
Administrivia

- No smoking
- Homework 1 due next Thursday 11:59 p.m.
  - Everyone must read and sign our collaboration policy
- First reading assignment due Tuesday
  - Effective Java Items 15 and 16

Key concepts from Tuesday

- Introduction to this course (17-214)
  - Object-oriented programming (via Java)
  - Design
  - Design
  - Design
  - Concurrency

Key concepts from Tuesday



https://travis-ci.org/images/logos/TravisCI-Mascot-1.png
https://www.unixmen.com/use-git-commands-linux-terminal/
https://bit.ly/2ZAnuPf\
https://github.githubassets.com/images/modules/logos_page/Octocat.png
https://junit.org/junit4/images/junit5-banner.png

Java is everywhere

TIOBE Index for August 2019

August Headline: Silly season in the programming language world

Nothing much has changed during July in the TIOBE index. In the top 10 only Objective-C and SQL have swapped positions. We need a magnifying glass to see some other noteworthy changes: Rust went from #33 to #28, TypeScript from #41 to #35 and Julia from #50 to #39. It is also interesting to note that Kotlin doesn't seem to come closer to the top 20. This month it even lost 2 positions: from #43 to #45.

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found here.

Aug 2019	Aug 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.028%	-0.85%
2	2		C	15.154%	+0.19%
3	4	^	Python	10.020%	+3.03%
4	3	v	C++	6.057%	-1.41%
5	6	^	C#	3.842%	+0.30%

https://www.tiobe.com/tiobe-index/

Outline

I. Hello World!
II. The type system
III. Quick ‘n’ dirty I/O
IV. Collections
V. Methods common to all Objects

institute for SOFTWARE RESEARCH

The “simplest” Java Program

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

The “simplest” Java Program

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

Complication: You must use a `class` even if you aren’t doing OO programming.

The "simplest" Java Program

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

- Every application must provide a `main` method
- Entry point to the program
- Always "`public static void main`"

The “simplest” Java Program

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

Return type

Indicates whether method is shared by whole class or is different for each instance.

Specifies who can “see” the method.

The “simplest” Java Program

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

Complication: `main` must declare command-line arguments even if it doesn’t use them.

isr institute for SOFTWARE RESEARCH

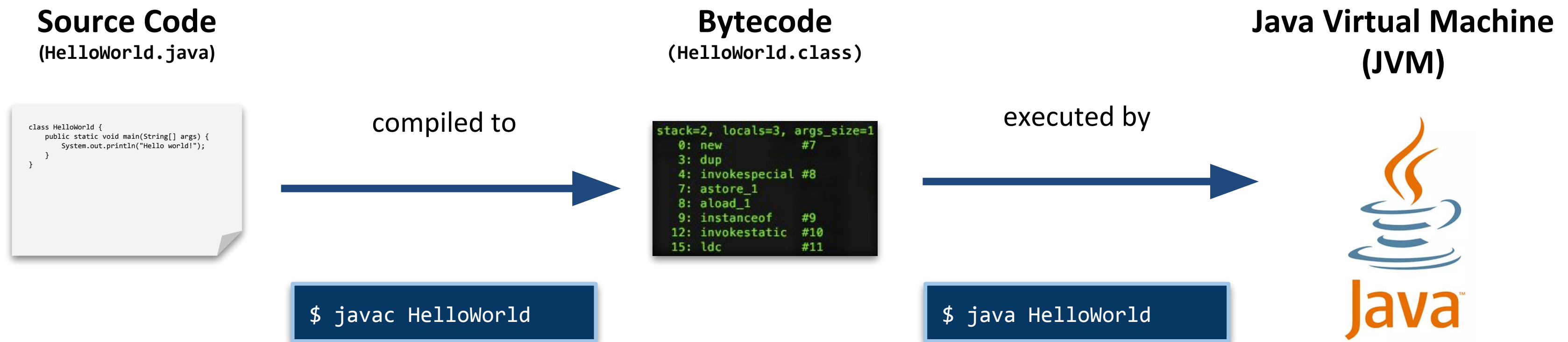
The “simplest” Java Program

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

Uses the `System` class from the core library to print "Hello world!" to standard output (console).

institute for SOFTWARE RESEARCH

Execution is a bit complicated



http://images.slideplayer.com/21/6322821/slides/slide_9.jpg
https://www.jonbell.net/wp-content/uploads/2015/10/Screen-Shot-2015-10-08-at-2.16.41-PM.png
https://www.theverge.com/2012/10/18/3524036/os-x-update-removes-java-web-browsers

It's not all so bad!

- Has many good points to balance shortcomings
- Some verbosity is not a bad thing
  - Can reduce errors and increase readability
- Modern IDEs eliminate much of the pain
  - Type `psvm` instead of `public static void main`
- Managed runtime has many advantages
  - Safe, flexible, enables garbage collection
- It may not be best language for Hello World…
  - But Java is very good for large-scale programming!

Outline

I. Hello World!
II. The type system
III. Quick ‘n’ dirty I/O
IV. Collections
V. Methods common to all objects

isr institute for SOFTWARE RESEARCH

Java type system

Primitive Types

```
int     long     short
char    boolean  byte
float   double
```

- **Dirt cheap**
- **On stack, exist only when used**
- **No identity other than value**

Object Reference Types

Classes, interfaces, arrays, enums, annotations, strings, exceptions

- **More expensive**
- **On heap, garbage collected**
- **Identity is distinct from value**

True or false?

```
int x = 5;
int y = 5;
System.out.println(x == y);
---------------------------
```

```
String x = “foo”;
String y = x;
System.out.println(x == y);
---------------------------
```

```
String x = “foo”;
String y = “foo”;
System.out.println(x == y);
---------------------------
```

True or false?

```
int x = 5;
int y = 5;
System.out.println(x == y);
---------------------------
true
```

```
String x = “foo”;
String y = x;
System.out.println(x == y);
---------------------------
```

```
String x = “foo”;
String y = “foo”;
System.out.println(x == y);
---------------------------
```

True or false?

```
int x = 5;
int y = 5;
System.out.println(x == y);
---------------------------
true
```

```
String x = “foo”;
String y = x;
System.out.println(x == y);
---------------------------
true
```

```
String x = “foo”;
String y = “foo”;
System.out.println(x == y);
---------------------------
```


isr institute for SOFTWARE RESEARCH

True or false?

```
int x = 5;
int y = 5;
System.out.println(x == y);
---------------------------
true
```

```
String x = “foo”;
String y = x;
System.out.println(x == y);
---------------------------
true
```

```
String x = “foo”;
String y = “foo”;
System.out.println(x == y);
---------------------------
false
```


isr institute for SOFTWARE RESEARCH

Identity vs. value

`x == y` compares the *identity* of x and y

- for primitives: identity = value
- for objects: identity = address on the heap

`x.equals(y)` compares the *contents* of x and y

```
String x = “foo”;
String y = “foo”;
System.out.println(x == y); // false
System.out.println(x.equals(y)); // true
```


isr institute for SOFTWARE RESEARCH

Primitive types

- `int` 32-bit signed integer
- `long` 64-bit signed integer
- `byte` 8-bit signed integer
- `short` 16-bit signed integer
- `char` 16-bit unsigned integer/character
- `float` 32-bit IEEE 754 floating point number
- `double` 64-bit IEEE 754 floating point number
- `boolean` Boolean value: `true` or `false`

institute for SOFTWARE RESEARCH

Warning: Deficient primitive types

- `byte`, `short` – use `int` instead!
  - `byte` is broken – should have been unsigned
- `float` – use `double` instead!
  - Provides too little precision
- Only compelling use case is large arrays, especially in resource-constrained environments

isr institute for SOFTWARE RESEARCH

Objects

- All non-primitives are represented by objects.
- An **object** is a bundle of state and behavior
- State – the data contained in the object
  - In Java, these are the **fields** of the object
- Behavior – the actions supported by the object
  - In Java, these are called **methods**
  - Method is just OO-speak for function
  - Invoke a method = call a function

Classes

- Every object has a class
  - A class defines methods and fields
  - Methods and fields collectively known as **members**
- Class defines both type and implementation
  - Type ≈ where the object can be used
  - Implementation ≈ how the object does things
- Loosely speaking, the methods of a class are its **Application Programming Interface (API)**
  - Defines how users interact with its instances

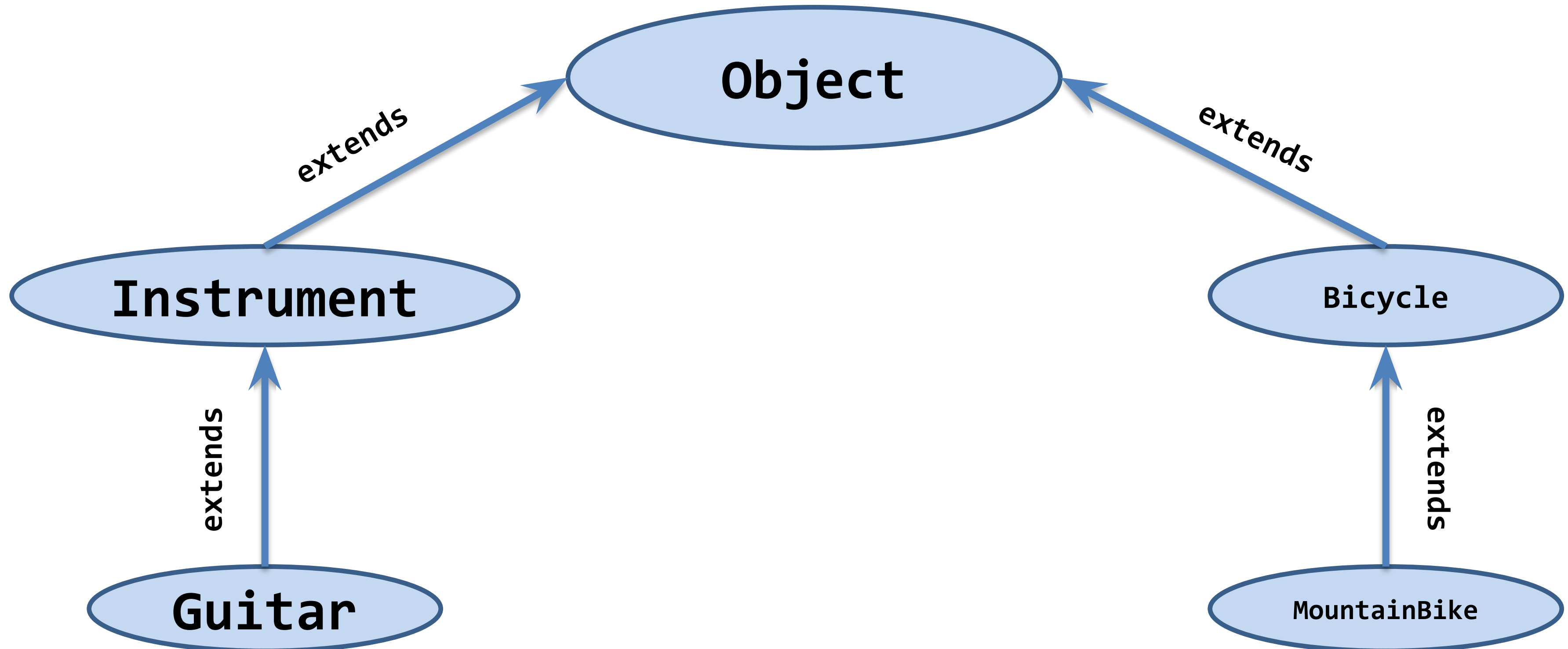
isr institute for SOFTWARE RESEARCH

The class hierarchy

The root is Object

All classes except `Object` have one parent class

A class is an instance of all its superclasses



Implementation inheritance

- A class:
  - Inherits visible fields and methods from its superclasses
  - Can override methods to change their behavior
- Overriding method implementation must obey contract(s) of its superclass(es)
  - Ensures subclass can be used anywhere superclass can
  - Liskov Substitution Principle (LSP)

Interface types

- Defines a type without an implementation
- Much more flexible than class types
  - An interface can extend one or more others
  - A class can implement multiple interfaces

```
interface Comparator {
    boolean compare(int i, int j);
}
class AscendingComparator implements Comparator {
    public boolean compare(int i, int j) { return i < j; }
}
class DescendingComparator implements Comparator {
    public boolean compare(int i, int j) { return i > j; }
}
```

Java arrays

- Conceptually represented as an object
  - Provides `.length`, runtime bounds-checking

```
String[] answers = new String[42];
if (answers.length == 42) {
    answers[42] = "no"; // ArrayIndexOutOfBoundsException
}
```

Java enums

- Like C enumerations, but represented as an object
  - Provides many object-oriented features, type safety, …

```
enum Planet { MERCURY, VENUS, EARTH, MARS,
              JUPITER, SATURN, URANUS, NEPTUNE; }
```

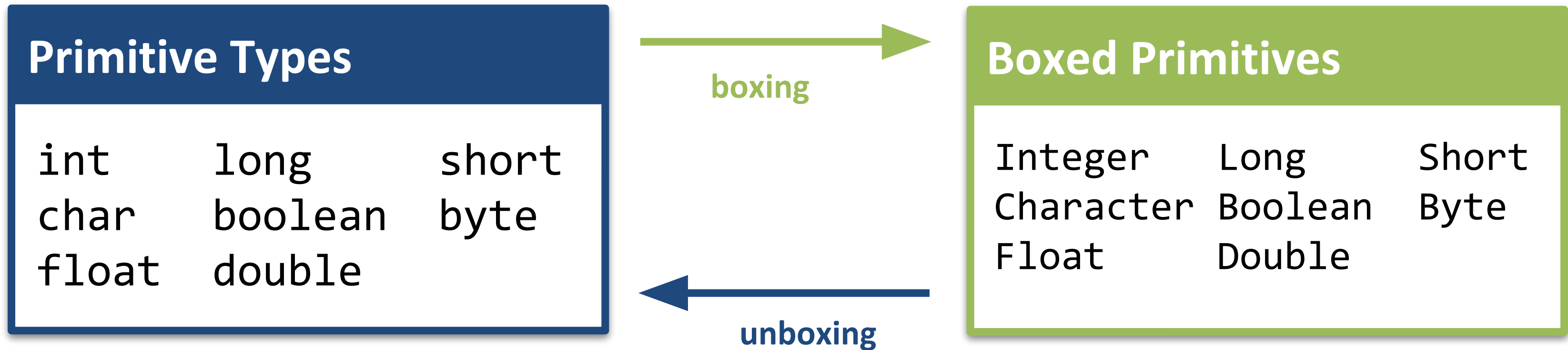
```
Planet location = …;
if (location.equals(Planet.EARTH)) {
    System.out.println("Honey, I'm home!");
}
```

Java annotations

- Annotations mark code without any immediate functional effect (e.g., `@Override, @Deprecated, @SuppressWarnings`)

```
class Bicycle {
    ...
    @Override
    public String toString() {
        return ...;
    }
}
```

Boxed primitives



- Allows primitives to be used in contexts requiring objects
  - canonical use case is collections (e.g., `HashSet<Integer>`)
- Don’t used boxed primitives unless you must!
- Language does autoboxing and auto-unboxing
  - blurs but does not remove distinction -- use carefully!

isr institute for SOFTWARE RESEARCH

Prefer primitives to boxed primitives

```
public class BoxOfTricks {
    public static Integer n;

    public static void main(String [] args) {
        if (n == 0)
            System.out.println(“That looks okay?”);
        else
            System.out.println(“I think not.”);
    }
}
```

Prefer primitives to boxed primitives

```
public class BoxOfTricks {
    public static Integer n = null;

    public static void main(String [] args) {
        if (n == 0) // throws NullPointerException
            System.out.println("That looks okay?");
        else
            System.out.println("I think not.");
    }
}
```

For more examples, see Effective Java, Item 61.

What does this fragment print?

```
int[] a = new int[]  { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

int i;
int sum1 = 0;
for (i = 0; i < a.length; i++) {
    sum1 += a[i];
}

int j;
int sum2 = 0;
for (j = 0; i < a.length; j++) {
    sum2 += a[j];
}

System.out.println(sum1 - sum2);
```

Maybe not what you expect!

```
int[] a = new int[]  { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

int i;
int sum1 = 0;
for (i = 0; i < a.length; i++) {
    sum1 += a[i];
}
int j;
int sum2 = 0;
for (j = 0; i < a.length; j++) { //  Copy/paste error!
    sum2 += a[j];
}
System.out.println(sum1 - sum2);
```

You might expect it to print 0, but it prints 55

institute for SOFTWARE RESEARCH

You could fix it like this…

```
int[] a = new int[]  { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

int i;
int sum1 = 0;
for (i = 0; i < a.length; i++) {
    sum1 += a[i];
}
int j;
int sum2 = 0;
for (j = 0; j < a.length; j++) {
    sum2 += a[j];
}

System.out.println(sum1 - sum2);  // Now prints 0, as expected
```

But this fix is far better…

```
int sum1 = 0;
for (int i = 0; i < a.length; i++) {
    sum1 += a[i];
}

int sum2 = 0;
for (int i = 0; i < a.length; i++) {
    sum2 += a[i];
}

System.out.println(sum1 - sum2);  // Prints 0
```

- Reduces scope of index variable to loop
- Shorter and less error prone

This fix is better still!

```
int sum1 = 0;
for (int x : a) {
   sum1 += x;
}

int sum2 = 0;
for (int x : a) {
   sum2 += x;
}

System.out.println(sum1 - sum2); // Prints 0
```

- Eliminates scope of index variable **entirely!**
- Even shorter and less error prone

isr institute for SOFTWARE RESEARCH

Lessons from the quiz

- **Minimize scope of local variables** [EJ Item 57]
  - Declare variables at point of use
- Initialize variables in declaration
- Use common idioms
- Watch out for *bad smells in code*
  - Such as index variable declared outside loop

Outline

I. Hello World!
II. The type system
III. Quick ‘n’ dirty I/O
IV. Collections
V. Methods common to all objects

Output

- Unformatted

```
System.out.println("Hello World");
System.out.println("Radius: " + r);
System.out.println(r * Math.cos(theta));
System.out.println();
System.out.print("*");
```

Aside: Overloaded Methods

```
System.out.println(r * Math.cos(theta));
System.out.println();
```

`void`	`println()` Terminates the current line by writing the line separator string.
`void`	`println(boolean x)` Prints a boolean and then terminate the line.
`void`	`println(char x)` Prints a character and then terminate the line.
`void`	`println(char[] x)` Prints an array of characters and then terminate the line.
`void`	`println(double x)` Prints a double and then terminate the line.
`void`	`println(float x)` Prints a float and then terminate the line.
`void`	`println(int x)` Prints an integer and then terminate the line.
`void`	`println(long x)` Prints a long and then terminate the line.
`void`	`println(Object x)` Prints an Object and then terminate the line.
`void`	`println(String x)` Prints a String and then terminate the line.

Output

- Unformatted

```
System.out.println("Hello World");
System.out.println("Radius: " + r);
System.out.println(r * Math.cos(theta));
System.out.println();
System.out.print("*");
```

- Formatted

```
System.out.printf("Radius: %d%n", r); // better!
System.out.printf("%d * %d = %d%n", a, b, a * b); // Varargs
```


institute for SOFTWARE RESEARCH

Command line input example

Echoes all command line arguments

```
class Echo {
    public static void main(String[] args) {
        for (String arg : args) {
            System.out.print(arg + " ");
        }
    }
}

$ java Echo The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
```

Command line input with parsing

Prints GCD of two command line arguments

```
class Gcd {
    public static void main(String[] args) {
        int i = Integer.parseInt(args[0]);
        int j = Integer.parseInt(args[1]);
        System.out.println(gcd(i, j));
    }

    static int gcd(int i, int j) {
        return i == 0 ? j : gcd(j % i, i);
    }
}

$ java Gcd 11322 35298
666
```


isr institute for SOFTWARE RESEARCH

Scanner input

Counts the words on standard input

```
class Wc {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        long result = 0;
        while (sc.hasNext()) { // whitespace delimiter
            sc.next(); // consume token
            result++;
        }
        System.out.println(result);
    }
}

$ java Wc < Wc.java
32
```

Outline

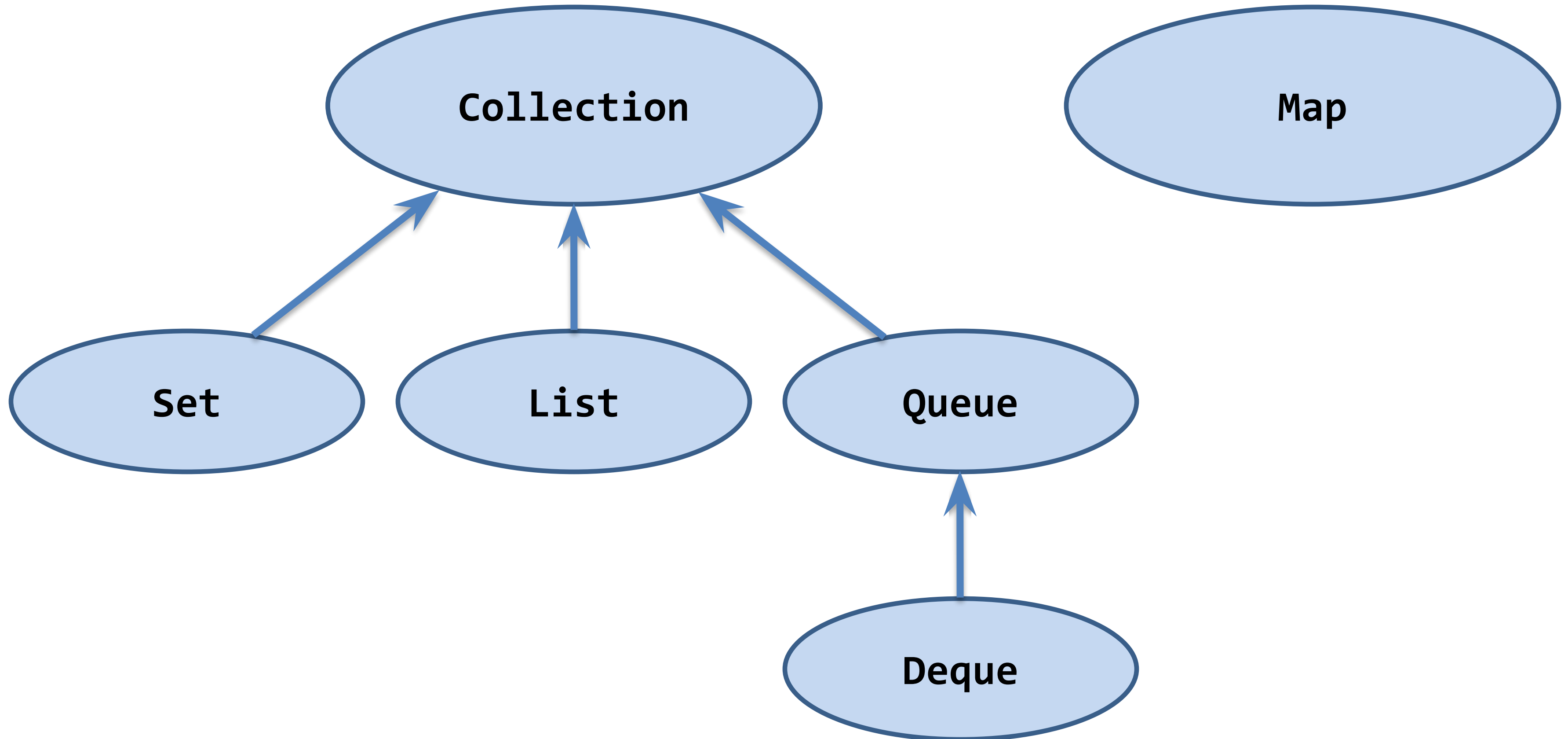
I. Hello World!
II. The type system
III. Quick ‘n’ dirty I/O
IV. Collections
V. Methods common to all objects

isr institute for SOFTWARE RESEARCH

Java Collections

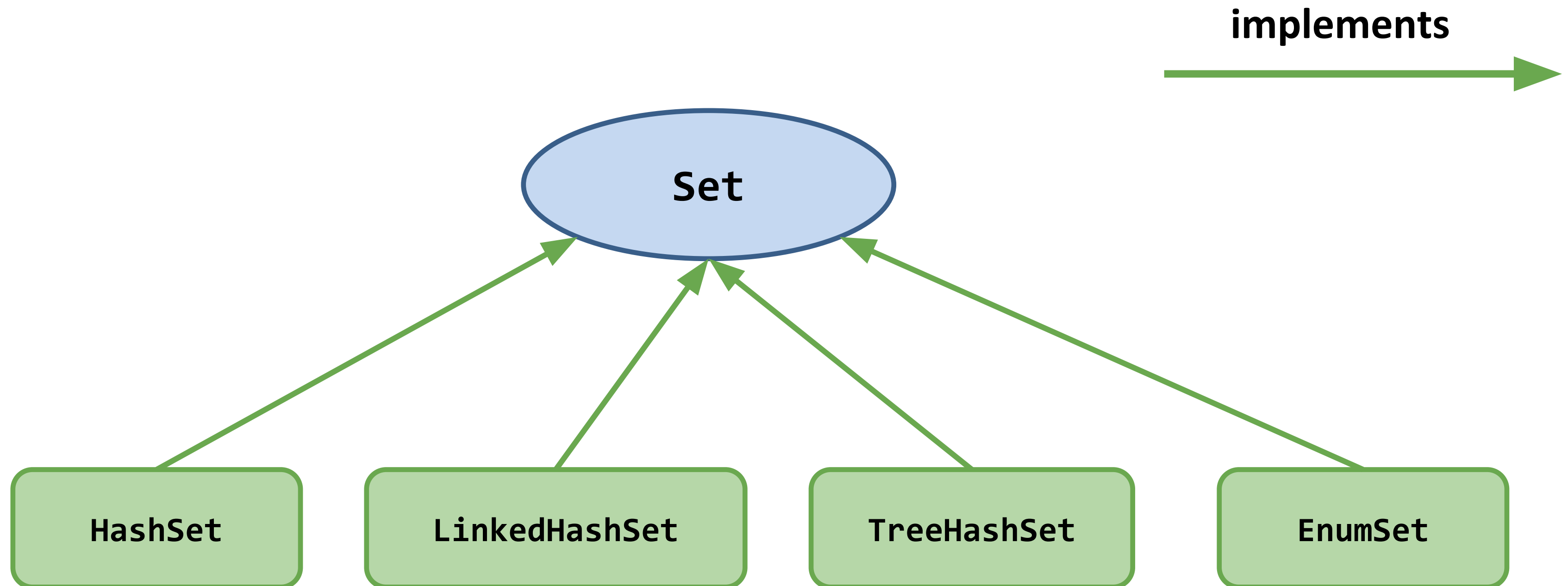
- A collection is an object that represents a group of objects.
- **Java Collections Framework**:
  - Interfaces for common abstract data structures
  - Classes that implement those data structures
  - Includes **algorithms** (e.g., searching, sorting).
    - algorithms are *polymorphic*: can be used on many different implementations of collection interfaces.

Primary collection interfaces



institute for SOFTWARE RESEARCH

Implementations of Set



isr institute for SOFTWARE RESEARCH

Collections usage example 1

Squeeze duplicate words out of command line

```
public class Squeeze {
   public static void main(String[] args) {
       Set<String> s = new LinkedHashSet<>();
       for (String word : args)
           s.add(word);
       System.out.println(s);
   }
}

$ java Squeeze I came I saw I conquered
[I, came, saw, conquered]
```


institute for SOFTWARE RESEARCH

Collections usage example 2

Print unique words in lexicographic order

```
public class Lexicon {
   public static void main(String[] args) {
       Set<String> s = new TreeSet<>();
       for (String word : args)
           s.add(word);
       System.out.println(s);
   }
}

$ java Lexicon I came I saw I conquered
[I, came, conquered, saw]
```

Collections usage example 3

Print index of first occurrence of each word

```
class Index {
    public static void main(String[] args) {
        Map<String, Integer> index = new TreeMap<>();

        // Iterate backwards so first occurrence wins
        for (int i = args.length - 1; i >= 0; i--)
            index.put(args[i], i);

        System.out.println(index);
    }
}

$ java Index if it is to be it is up to me to do it
{be=4, do=11, if=0, is=2, it=1, me=9, to=3, up=7}
```

Warning: Arrays are not collections

- Arrays and collections don't mix
  - If you try to mix them and get compiler warnings, take them seriously
- Generally speaking, prefer collections to arrays
  - See *Effective Java* Item 28 for details

More information on collections

- For *much* more information on collections, see the annotated outline:

  https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/doc-files/coll-index.html

- For more info on *any* library class, see javadoc
  - Search web for <fully qualified class name> 12
  - e.g., `java.util.scanner 12`

Outline

I. Hello World!
II. The type system
III. Quick ‘n’ dirty I/O
IV. Collections
V. Methods common to all objects

isr institute for SOFTWARE RESEARCH

Methods common to all objects

- How do collections know how to test objects for **equality**?
- How do they know how to **hash** and **print** them?

- The relevant methods are all present on `Object`
  - `equals` - returns true if the two objects are “equal”
  - `hashCode` - returns an `int` that must be equal for equal objects, and is likely to differ on unequal objects
  - `toString` - returns a printable string representation

`Object` implementations

- Provide *identity semantics*
  - `equals(Object o)` - returns `true` if o refers to this object
  - `hashCode()` - returns a near-random `int` that never changes over the object lifetime
  - `toString()` - returns a nasty looking string consisting of the type and hash code
    - For example: `java.lang.Object@659e0bfd`

isr institute for SOFTWARE RESEARCH

Overriding `Object` implementations

- No need to override `equals` and `hashCode` if you want identity semantics
  - When in doubt, don't override them
  - It's easy to get it wrong
- Nearly always override `toString`
  - `println` invokes it automatically
  - Why settle for ugly?

Overriding `toString`

Overriding `toString` is easy and beneficial

```
final class PhoneNumber {
    private final short areaCode;
    private final short prefix;
    private final short lineNumber;
      ...
    @Override public String toString() {
        return String.format("(%03d) %03d-%04d",
            areaCode, prefix, lineNumber);
    }
}

Number jenny = ...;
System.out.println(jenny);
```

Prints: `(707) 867-5309`

Summary

- Java is well suited to large programs; small ones may seem a bit verbose
- Bipartite type system – primitives & object refs
- A few simple I/O techniques will get you started
- Collections framework is powerful & easy to use