Computer Programming – CS 6011
Lecture 2: Java

Master of Software Development (MSD) Program

Fall 2023

Lecture 3 – Java

- Topics
  - Why Java
  - Java vs. C++
  - Junit testing
  - Files

Shifting to Java

- Great for OOP
  - Almost everything is an object
  - Garbage collector
- Portable language
- Java supports GUI programming
- Android adopted Java

In many ways, syntax of Java and C++ is very identical.

- Basic constructs:
  - If
  - Loops
  - Variables
  - Defining Classes
  - Writing Methods

Java vs C++

- All Java code is written in classes.
  - Even main:

```
public static void main(String args[])
{
   …
}
```

- All Java code goes in the **.java** file.
  - The name of the file must match the name of the class in the file – one class per file.

Cross Platform

- C++
  - .cpp – compiles to
  - .o – links to
  - executable
- Java
  - .java file – compiles to
  - .class file (Java Bytecode) – links to:
  - .jar file – which runs on Java Virtual Machine (JVM)
- JVM vs JRE vs JDK

Some syntax differences…

- Java uses *`import`* instead of *`#include`*.
- Java uses *`boolean`* instead of the shortened *`bool`*.
- Java uses the *`System.out.println()`* method instead of *`cout`*.
- Java uses the *`Scanner`* class (instead of *`cin`*). It can be hooked to any input stream.
  - `Scanner s = new Scanner( System.in );`
    `// s is a scanner hooked to the console`
  - `int i = s.nextInt(); // Like cin >> i;`
- Arrays:
  - `int[] myArray = new int[ size ]; // Very similar to C++.`
  - `myArray.length` provides the size of the array.

ArrayList

- Equivalent to std::vector in C++
- Dynamic data structure
- Access to elements by index

Example

```
ArrayList<Integer> aL=new ArrayList<Integer>();
 for ( int i = 1; i <= n; i++ ) {
      aL.add( i ); // to add elements // no []
 }
 aL.set(index,newVal ) // Update element at index - no []
 aL.get(index) // Get element at index - no []
```

Primitive/Object data types

- Example: ArrayList<Integer> aL=new ArrayList<Integer>();
- Integer or int? typo?
- Primitive data type vs Object data type (Wrapper class)

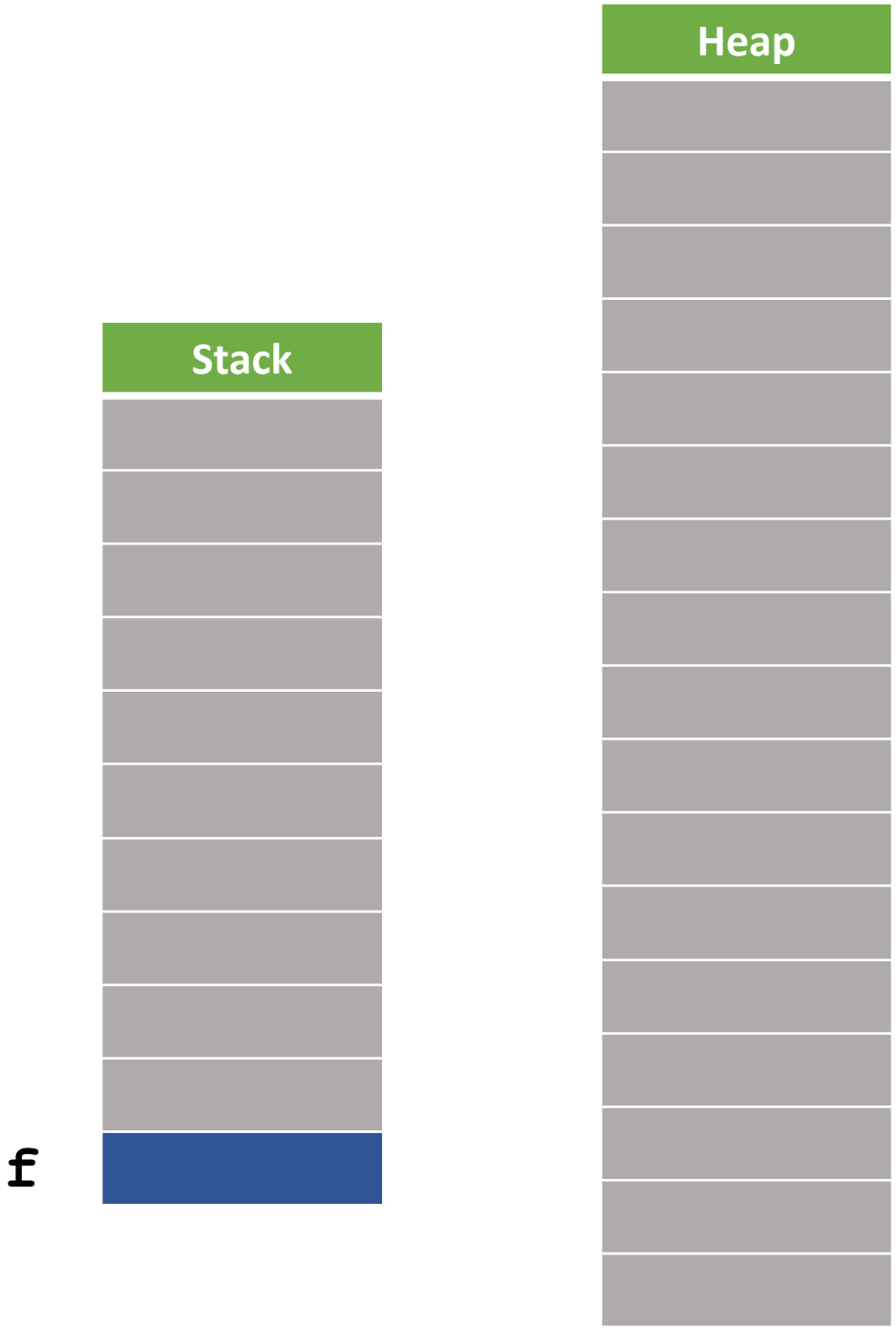
Primitive data type	Wrapper class
int	Integer
byte	Byte
short	Short
long	Long
float	Float
double	Double
char	Char
boolean	Boolean

Java vs C++ – More Syntax differences

- Object comparison
  - `MyClass s1, s2;`
  - `s1 == s2` //Typically not what you want
  - Usually you want: `s1.equals( s2 )`
  - String f1 = **new** String(**"ABC"**);
    String s2 = **new** String(**"ABC"**);
    System.***out***.println(s1==s2); //false
    System.***out***.println(s1.equals(s2)); //true

Object types in java are pointer types

- Objects are created on the heap
- A variable of Object type store the reference of that object
  - Just A pointer giving the address of that object in memory
  - Pointer in Java are not scary: no →, No &, No * , Just .

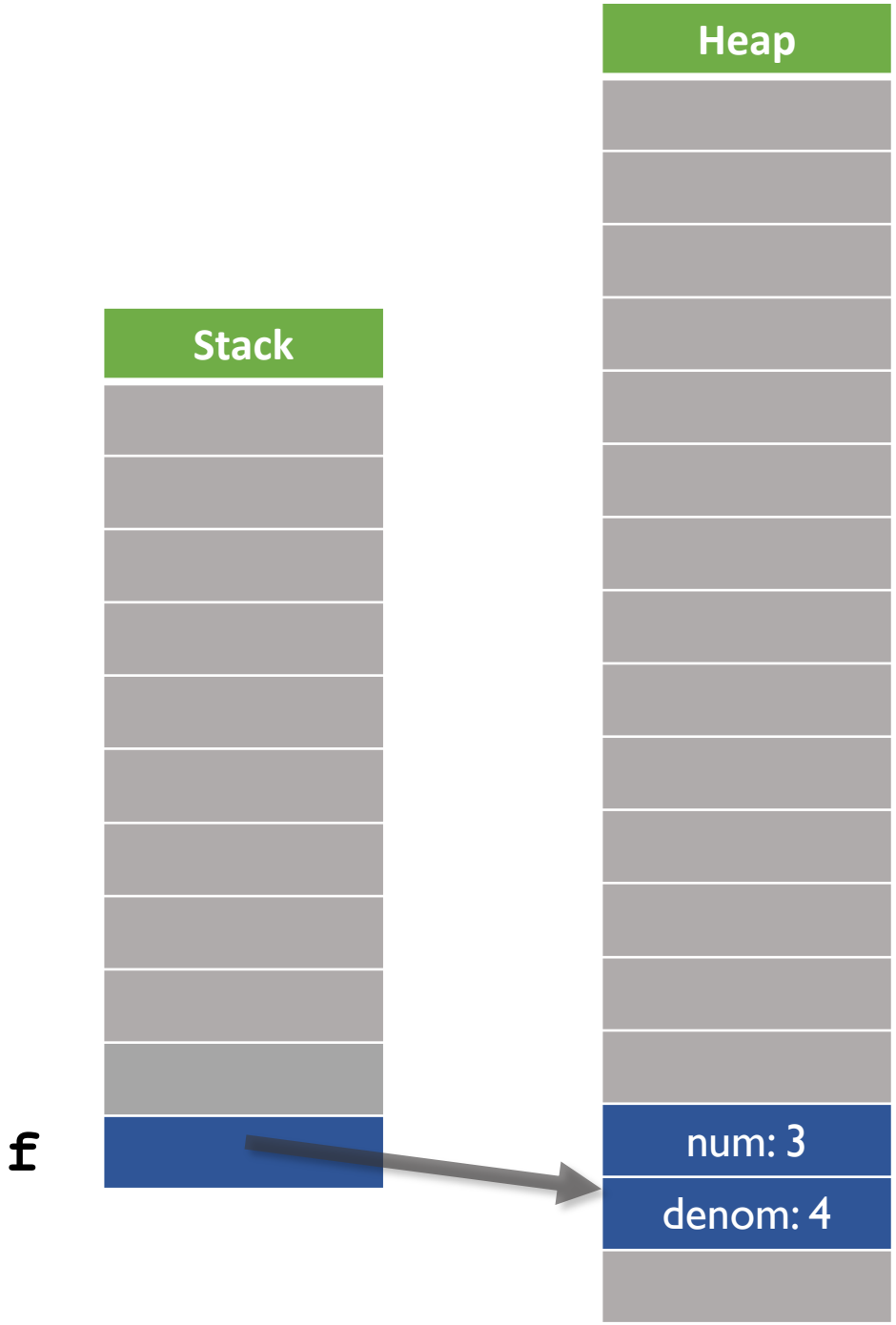


Java Object

```
Fraction f = new Fraction( 3, 4 );
```

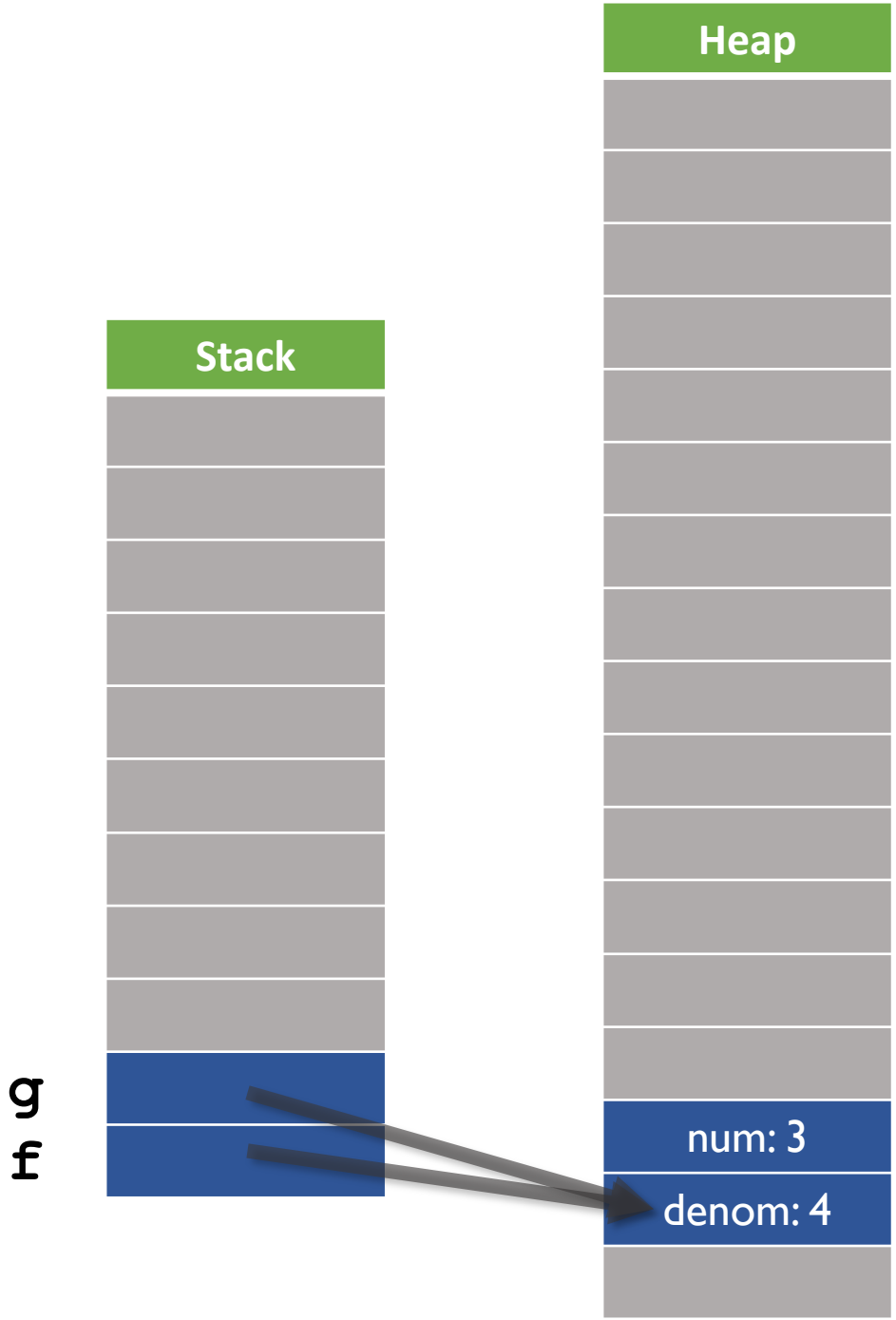
C++ Object

```
Fraction *f = new Fraction( 3, 4 );
```



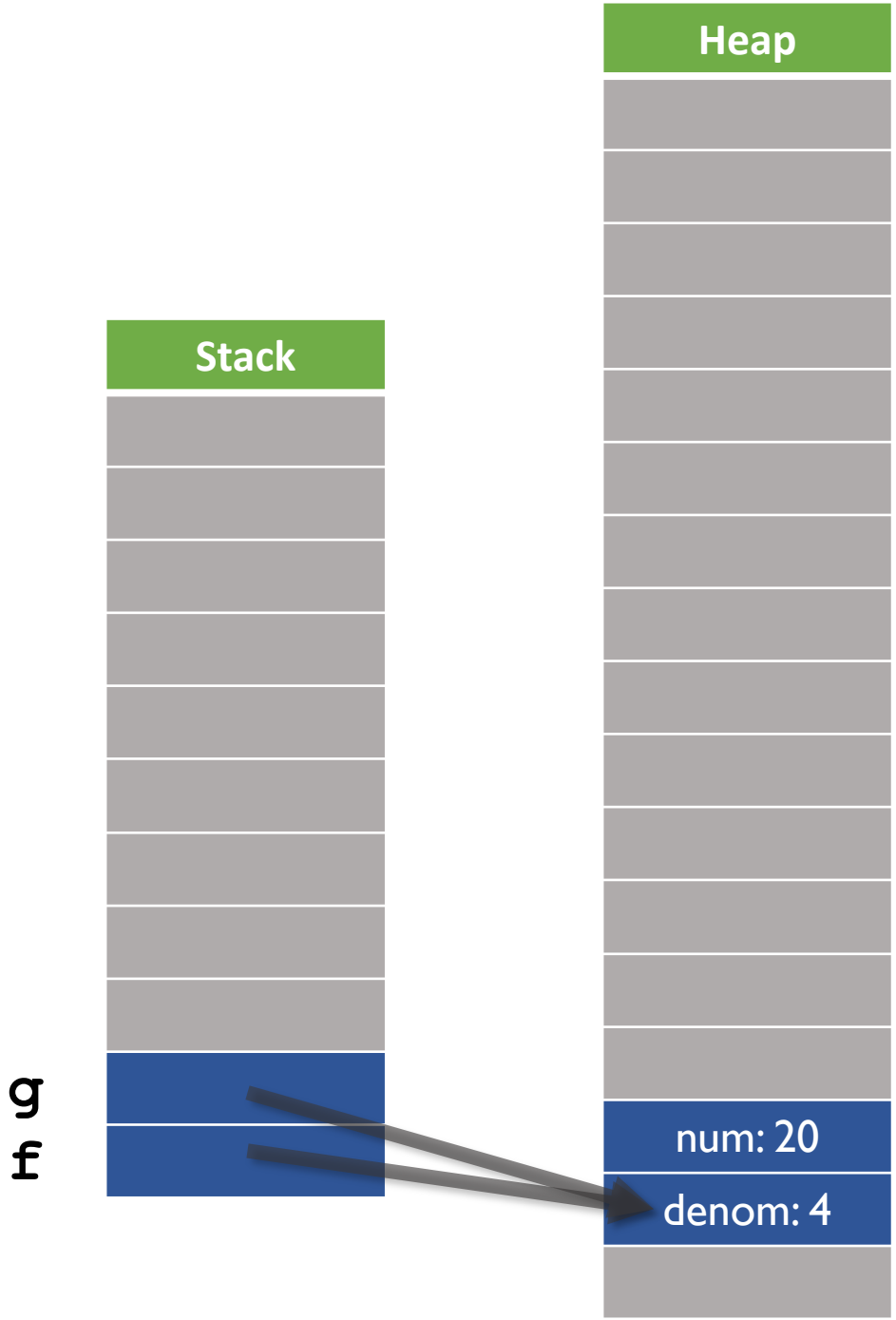
Java Object

```
Fraction f = new Fraction( 3, 4 );
   // Make space on the heap for a
   // Fraction Object
```



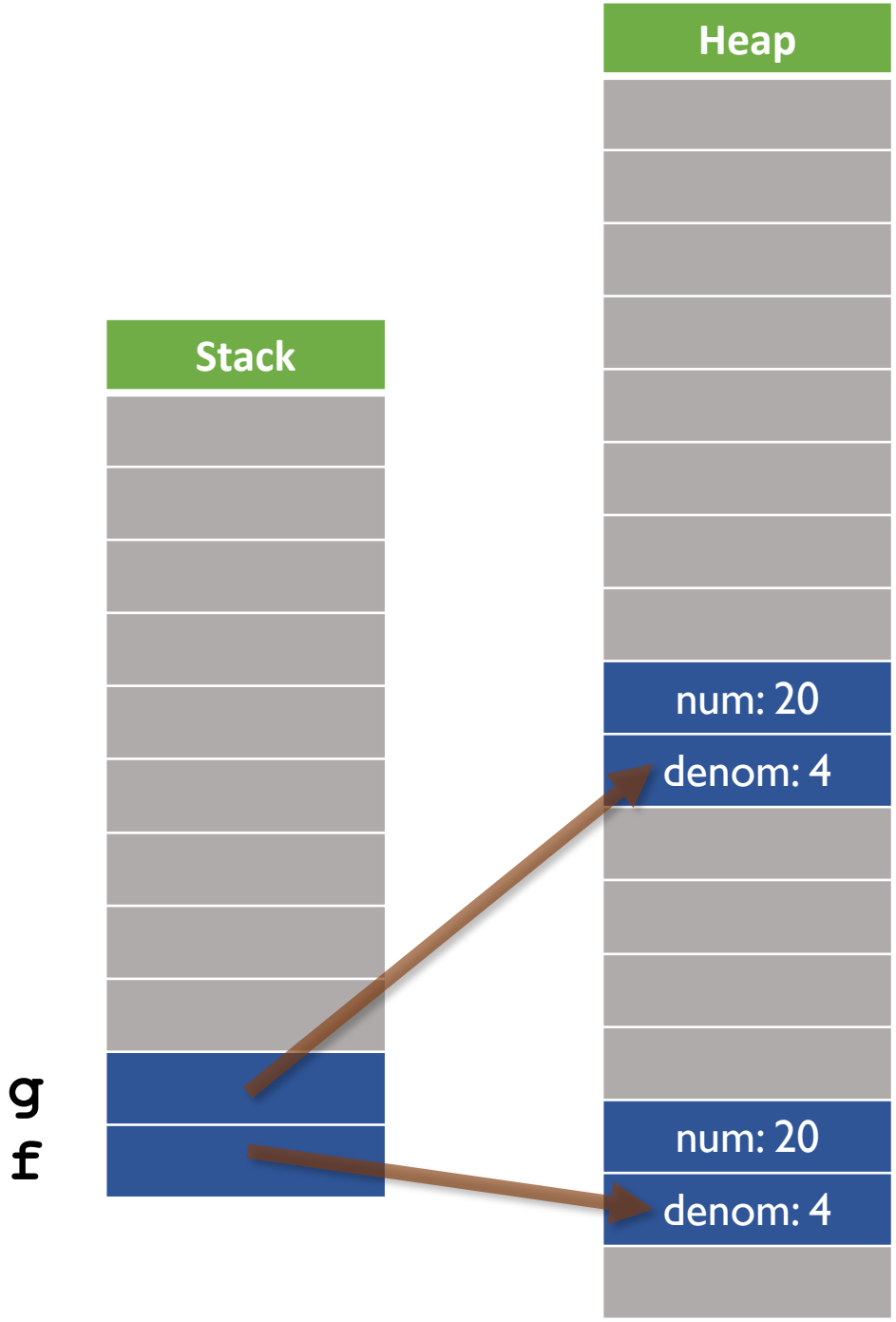
Copying a Java Object

```
Fraction f = new Fraction( 3, 4 );
  // Make space on the heap for a
  // Fraction Object
f.numerator_ // Access member var
f.reduce() // Call a method
Fraction g = f;
g.numerator = 20;
```



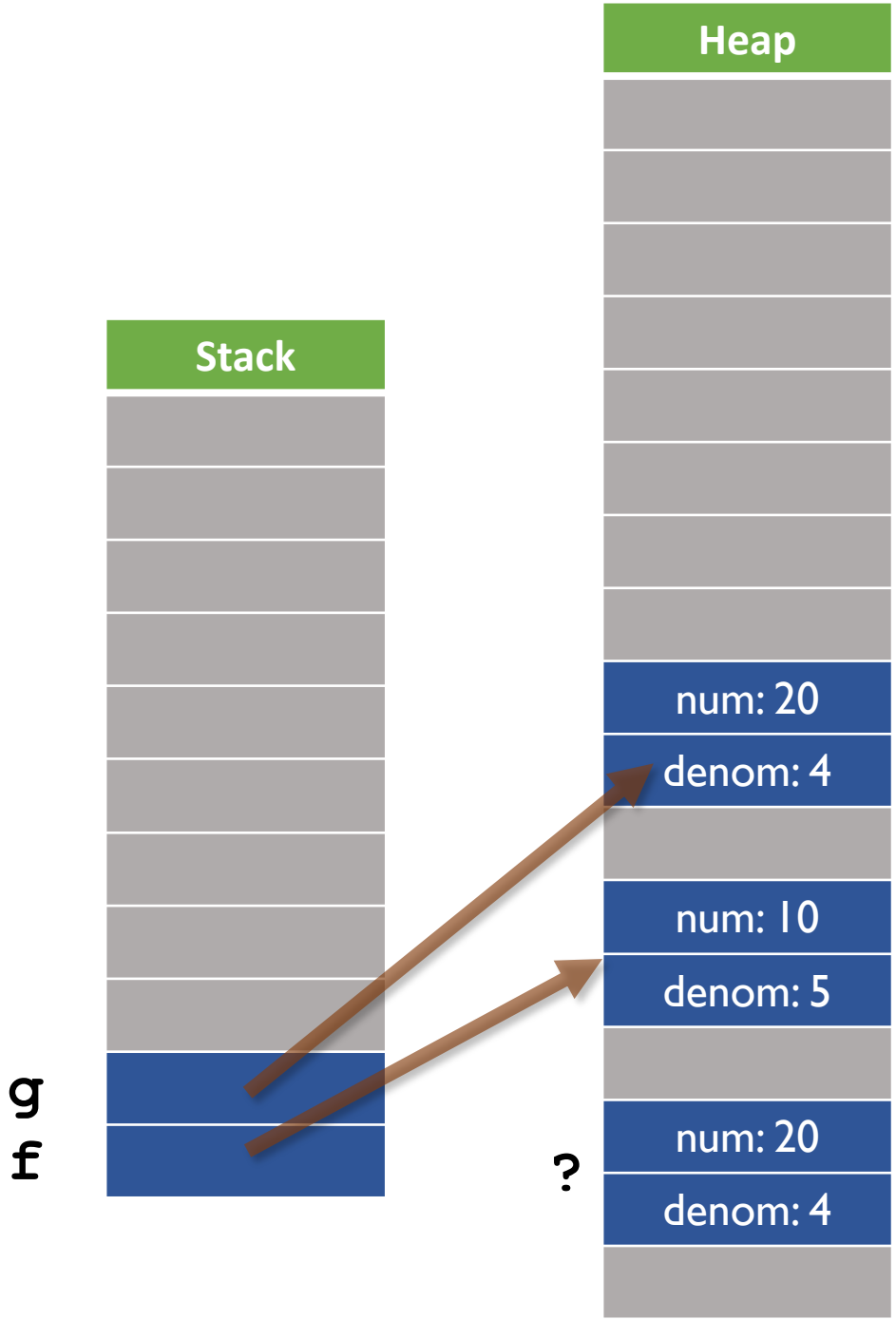
Copying a Java Object

```
Fraction f = new Fraction( 3, 4 );
  // Make space on the heap for a
  // Fraction Object
f.numerator_ // Access member var
f.reduce() // Call a method
Fraction g = f;
g.numerator = 20;
// f has also change.
```



Copying a Java Object

```
Fraction f = new Fraction( 3, 4 );
  // Make space on the heap for a
  // Fraction Object
f.numerator_ // Access member var
f.reduce() // Call a method
Fraction g = f;
f.numerator = 20;
// f has also change.
To create a new g:
Fraction g = new Fraction( f );
```



Memory leak

```
Fraction f = new Fraction( 3, 4 );
  // Make space on the heap for a
  // Fraction Object
f.numerator_ // Access member var
f.reduce() // Call a method
Fraction g = f;
g.numerator = 20;
// f has also change.
To create a new g:
Fraction g = new Fraction( f );
```

F = new Fraction (10,5);

Java Memory Management / Memory leaks

- All objects are created on the heap, and any variable referring to an object **is a pointer to that object**.
- In C++, `new`/`delete` to create and delete an object on the heap.
- In Java, pointers are handled like (and called) references to objects. To create an object, we use `new`.
- Java keeps track of all heap memory for you and will clean up any "leaked" memory using the GC. **No explicit object delete**.

Java vs C++

- High-level differences

	C++	**Java**
Platform	Platform-dependent	Platform-independent
Portability	Not portable	Portable
Memory Management	Manual	Automated (**Garbage Collector**)
Overloading	Operators and methods can be overloaded	Only method overloading
Pointers	Supports pointers explicitly	Supports pointers implicitly
Safety	?	Safer than C++

Java is Safer than C++

- Out of bounds memory default checks
- Java hides pointers to make the language safer.
  - Controls / limits dereferencing
  - No pointer arithmetic
  - No more delete
  - Without *delete* Java leaks memory all the time, so…
  - **Garbage Collection**
    - Java scans memory (perdiocally) to find non-accessible heap memory, and then automatically frees it (returns to the system)

Example Java File (and Class)

```
public class Puppy {
    private int puppyAge_; // Member Variables
    private String name_;
    public Puppy( String name ) { // This constructor has one parameter, name.
        System.out.println("Name chosen is :" + name );
        this.name_ = name;
        name_ = name;
        this.puppyAge_ = -1;
    }
    public void setAge( int age ) { puppyAge_ = age; }

    public int getAge( ) {
        System.out.println("Puppy's age is :" + puppyAge_ );
        return puppyAge_;
    }
    public static void main( String []args ){
        Puppy myPuppy = new Puppy("");
        myPuppy.setAge( 2 );
        int age = myPuppy.getAge( );
    }
}
```

Testing

- Right click your class and select Show Context Actions ==> Generate Test
  - Select OK (Press Fix if Junit library is not found in the module)
  - Make sure to pick Junit5 even if the default is Junit4
- Open NameTest.java
- Hover over (the error/red) *junit* in the import org.junit.jupiter.api.Assertions.*;
  - Add junit to the class path (if not added by IntelliJ).
- Add some test methods. See next slide.

Test Class Example:

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
class RectangleDemoTest {
    @Test
    public void perimeter() {
      int p1 = RectangleDemo.perimeter(2,2);

      // Qualify the assertEquals() with "Assertions." to say that it comes
      // from the Assertions library. The Assertions library, as can be seen
      // from the import above, is: org.junit.jupiter.api.Assertions.
      Assertions.assertEquals(p1, 8 );
    }
}
```

File Input / Output

```
File    file = new File( "test.txt" ); // Open file
// Attach file so scanner will read from it:
Scanner sc   = new Scanner( file );
// Reads an integer from the file.
int    i = sc.nextInt();
String s = sc.nextLine();  // Reads a line from the file
String s = sc.next(); // difference is that the delimiter is space
```

show case where nextLine could return empty string (end of line) ???

Tuesday Assignment(s)

- Lab – Java Hello World

- Assignment – Fractions and Rainfall in Java
  - Fractions due tomorrow, Rainfall due Thursday