# Formality of Java Programming Part 1

Steven Castellucci

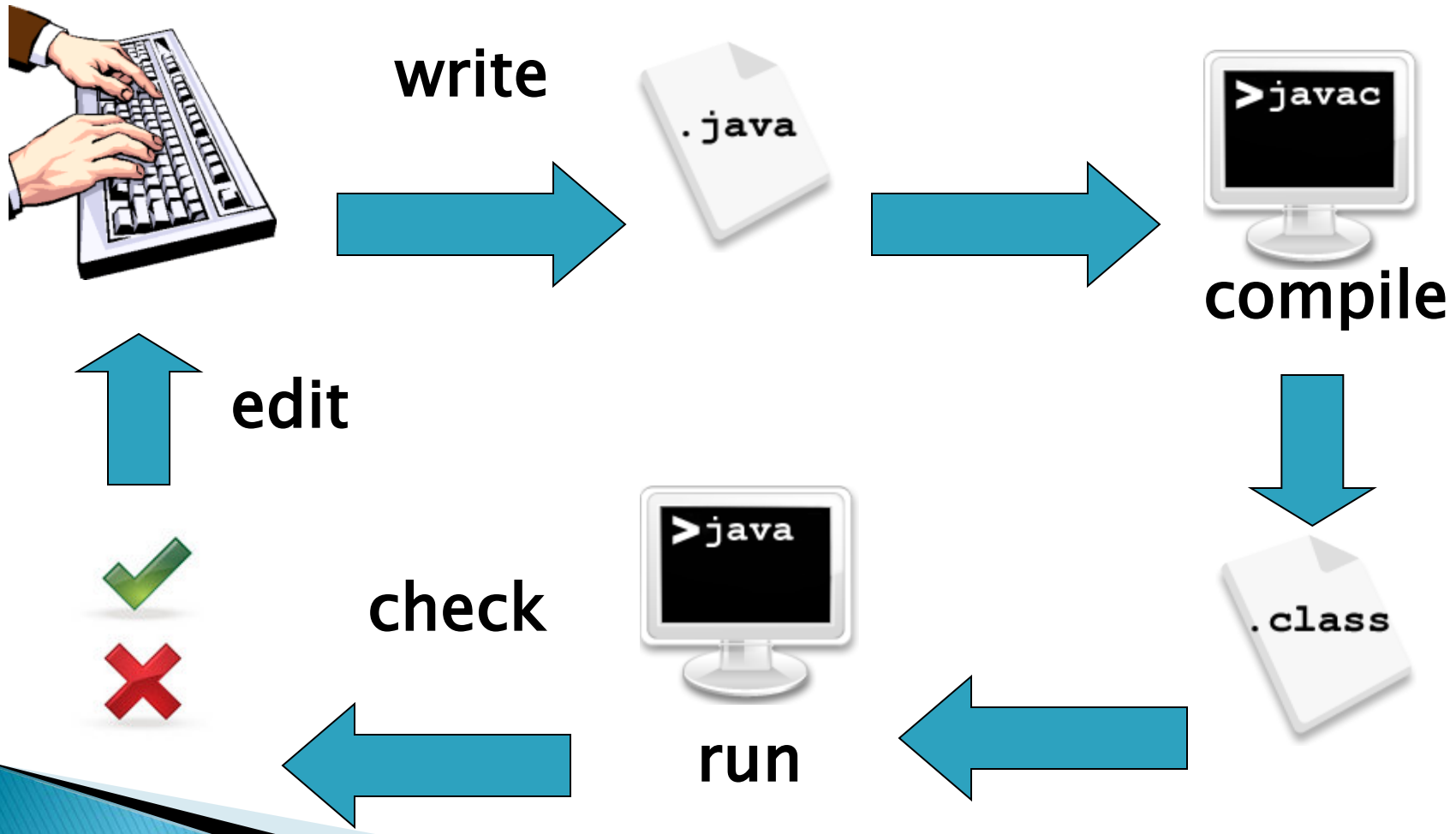
# Why Java?

- Syntax is similar to other languages
- Platform independent
- Widely used in industry

- Uses:
  - Server-side (Gmail backend)
  - Client-side (jEdit, Eclipse)
  - Mobile devices (Android*)
  - Consumer devices (Blu-ray players)

*Android programs use Java syntax and similar libraries.

# How to Program



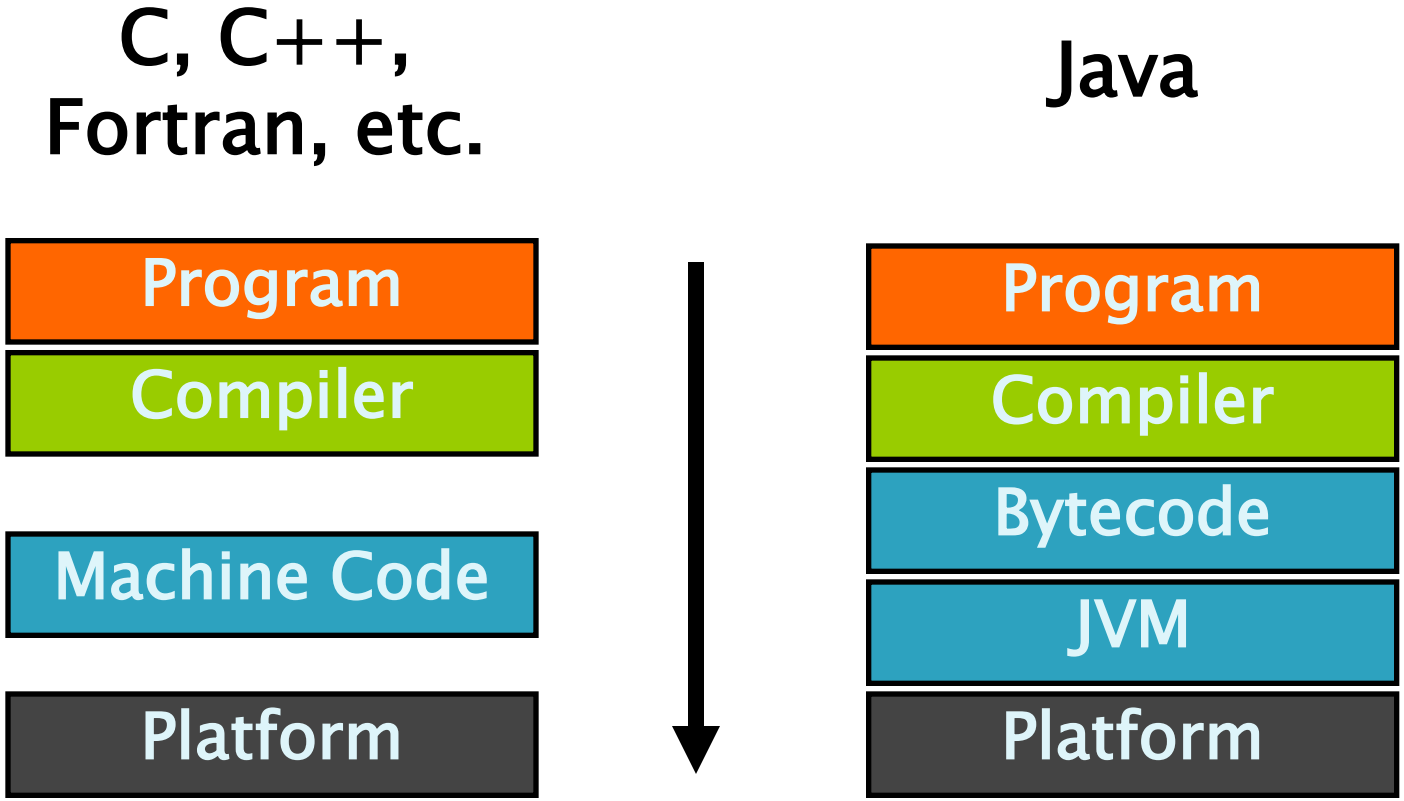
# How to Program

- Write code in a computer language (remember to save the file)
- Use the language's compiler to convert your code to machine-readable code
- Run your program
- Compare actual result to expected one
- Edit your code as necessary, and repeat

# Machine Language

- Why not just program in machine language?
- Machine languages are
  - Machine-dependant
  - Complex and verbose
  - Difficult to understand large programs
- Compilers abstract (i.e., remove) the complexities of machine language
- Programming languages simplify design

# Java Virtual Machine



# Keywords

- Have special meanings in Java
- Not to be used as identifiers, class names, etc.

| | | | | | |
|---|---|---|---|---|---|
| abstract | assert | | | | |
| boolean | break | byte | | | |
| case | catch | char | class | const | continue |
| default | do | double | | | |
| else | enum | extends | | | |
| final | finally | float | for | | |
| goto | | | | | |
| if | implements | import | instanceof | int | interface |
| long | | | | | |
| native | new | | | | |
| package | private | protected | public | | |
| return | | | | | |
| short | static | strictfp | super | switch | synchronized |
| this | throw | throws | transient | try | |
| void | volatile | | | | |
| while | | | | | |

# Logical Operators

- AND &&
- OR ||
- Equal ==
- Not equal !=
- Less than <
- Less than or equal <=
- Greater than >
- Greater than or equal >=

# Arithmetic Operators

- Add +
- Subtract -
- Multiply *
- Divide /
- Remainder %

# Primitive Data Types

- Integers
  - int (4 bytes): $[-2 \times 10^9 \,..\, 2 \times 10^9)$
  - long (8 bytes): $[-9 \times 10^{18} \,..\, 9 \times 10^{18})$
- Reals
  - float (4 bytes): $[-3.4 \times 10^{38} \,..\, 3.4 \times 10^{38})$, 7 sig. digits
  - double (8 bytes): $[-1.7 \times 10^{308} \,..\, 1.7 \times 10^{308})$, 15 sig. digits
- Characters
  - char (2 bytes): Unicode characters 0x0000 to 0xFFFF
- Boolean
  - boolean: (1byte): true or false

# Declaring Variables

- A variable's value can change during execution
- Declaration

  *primType identifier* = *value*;

  Where:

  *primType* is int, long, float, double, etc.
  *identifier* is the name you choose for the variable
  *value* is the value you want the variable to have
- Example

  int currentTemperature = 8;

# Declaring Constants

- A constant's value does not change during execution
- Declaration

  final *primType IDENTIFIER = value*;

  Where:

  *primType* is int, long, float, double, etc.

  *IDENTIFIER* is the name (all caps) you choose

  *value* is the value you want the constant to have
- Example

  final int INCHES_PER_FOOT = 12;

# Integer Division

- Satisfies the closure property
- When dividing an integer by another, the result in also an integer
- Ignore any remainder
- Don't confuse integer division with real division
  - Integer: $2 / 3 = 0$, $5 / 3 = 1$
  - Real: $2.0 / 3.0 = 2.0 / 3 = 2 / 3.0 = 0.66666\ldots$

# Operator Precedence

- Arithmetic operator precedence similar to order of operations learned in high school
  - Brackets
  - Multiplication and division (including remainder)
  - Addition and subtraction
- Left-to-right association
- Full details in Appendix 2

# Example

```
5 + (4 - 3) / 5 - 2 * 3 % 4
```

# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
```

# Example

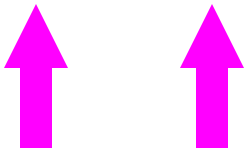
```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
```

# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
```

# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
```



# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
```

# Example

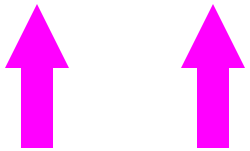
```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
```

# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
= 5 + 0 - 2
```

# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
= 5 + 0 - 2
```



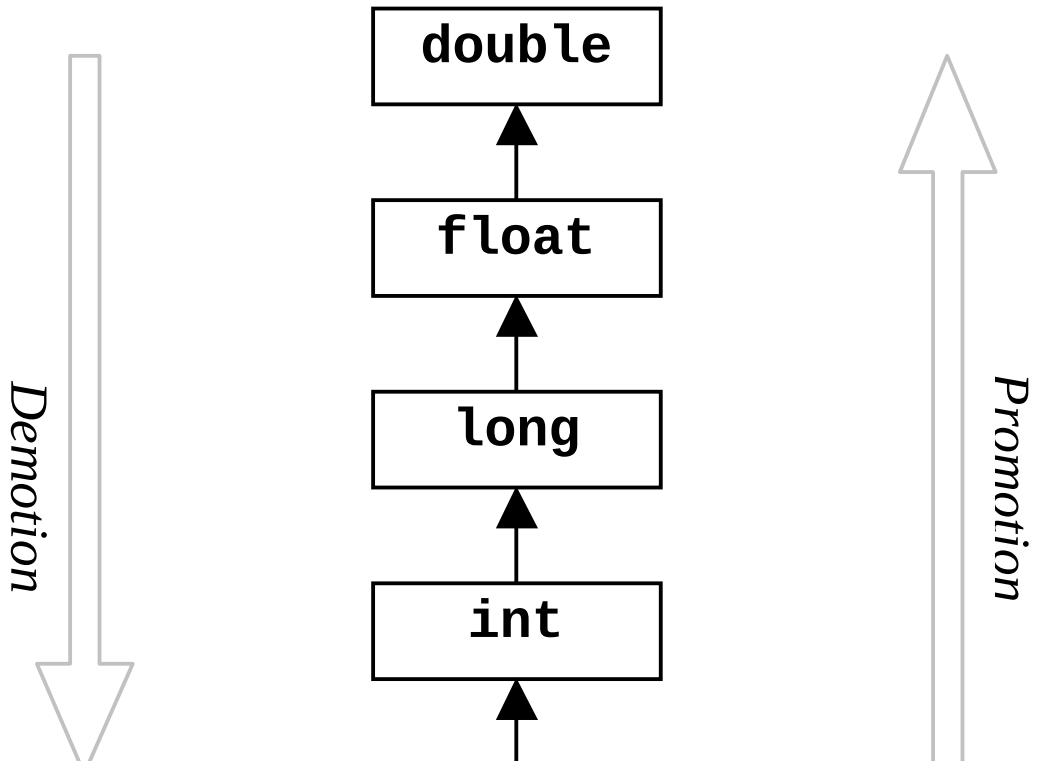
# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
= 5 + 0 - 2
= 5 - 2
```

# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
= 5 + 0 - 2
= 5 - 2
= 3
```

# Promotion



# Casting

- Returns compile-time exception

```
double aDbl = 5.0;
int bInt = 2;
int result = aDbl * bInt;
```

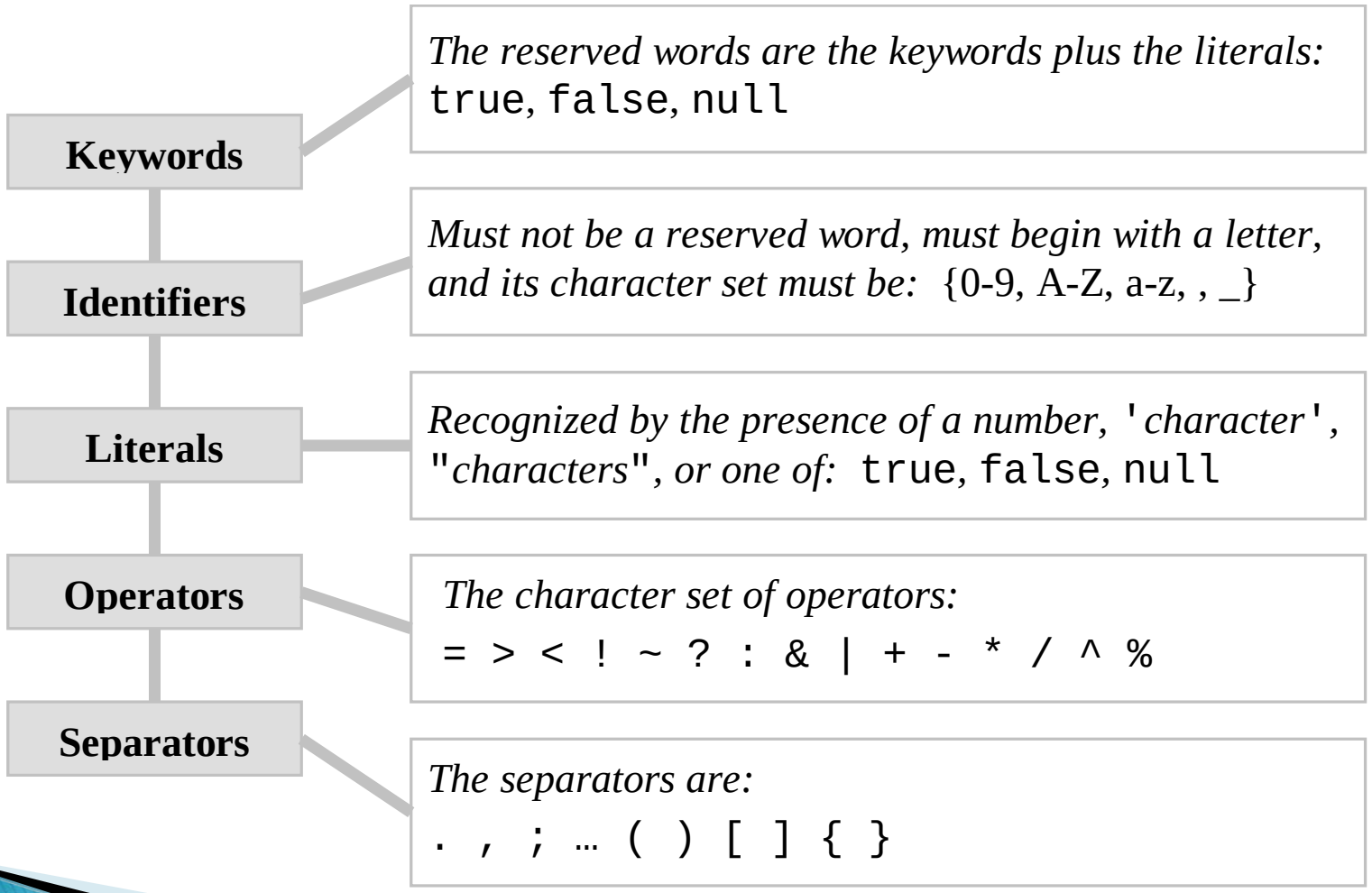
- Demotion accomplished with casting

```
double aDbl = 5.0;
int bInt = 2;
int result = (int) aDbl * bInt;
```

- Promotion via casting to force real division

  2 / 3 = 0, (double) 2 / 3 = 2 / (double) 3 = 0.66666…

# Lexical Elements



# Class template

```
// any needed package statement
// any needed import statements

public class SomeName
{
        // the attribute section
        // the constructor section
        // the method section
}
```

# Classes

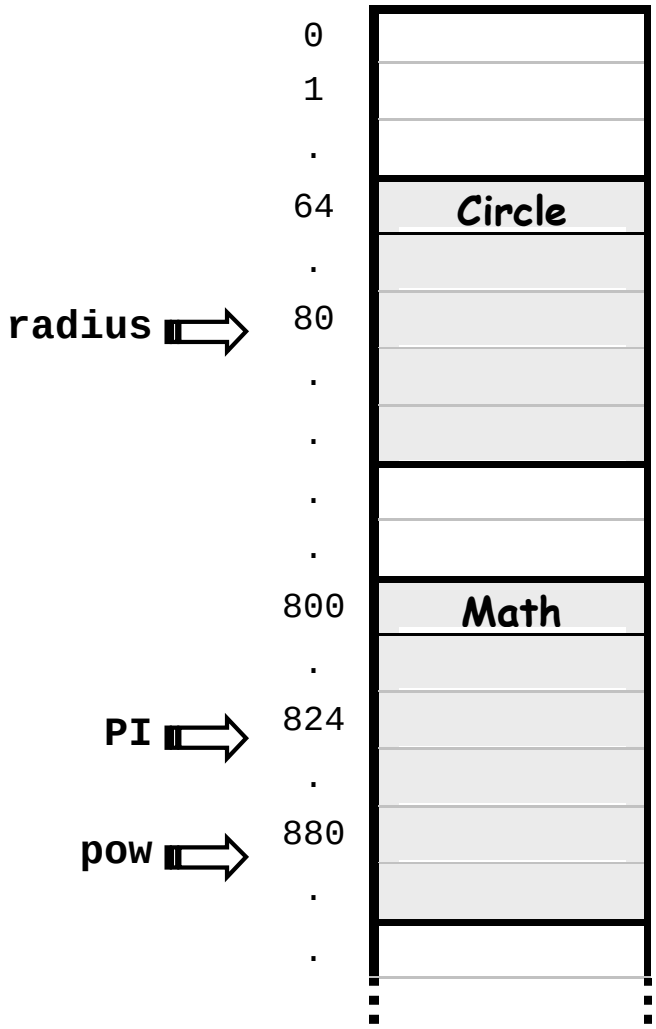
- A class can serve as…
  - An object factory, defining attributes and methods
  - A utility with static values and methods to perform calculations
  - A main class to orchestrate execution of an application

# Memory Diagrams

- The Circle class (below) uses an int field and a method in the Math utility class

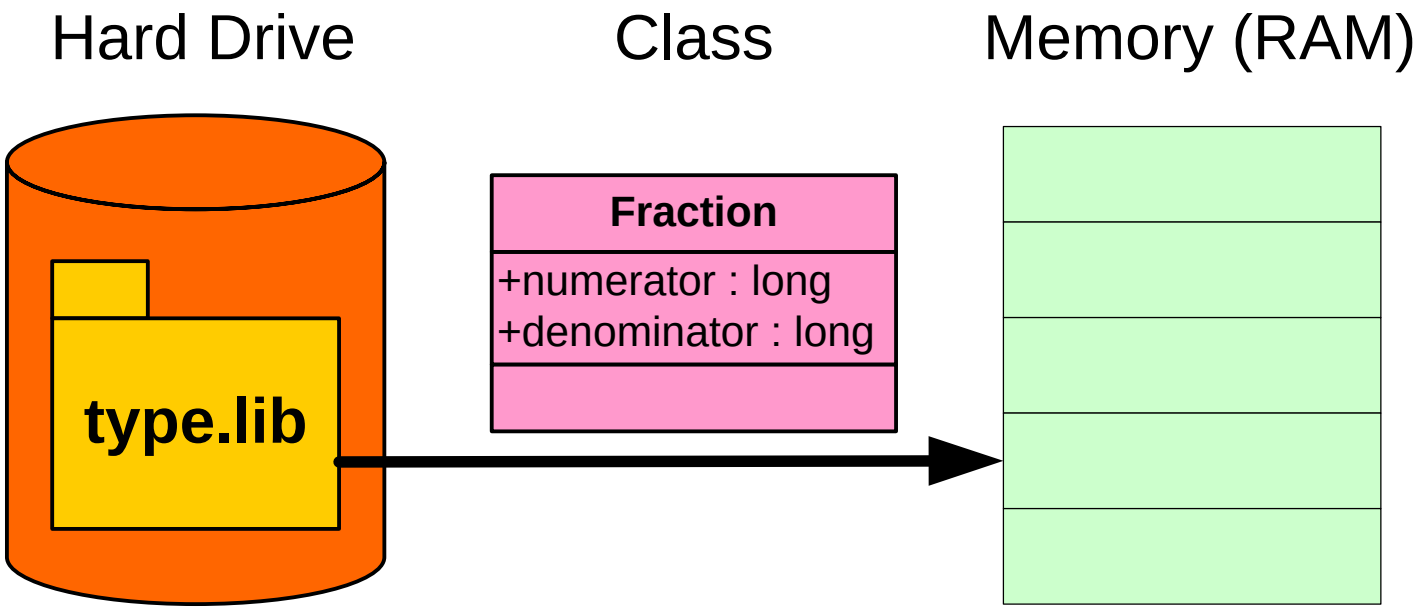
```
import java.util.Scanner;
import java.io.PrintStream;
public class Circle
{
   public static void main(String[] args)
   {
      Scanner input = new Scanner(System.in);
      PrintStream output = System.out;
      output.print("Enter radius: ");
      int radius = input.nextInt();
      output.println(Math.PI * Math.pow(radius, 2));
   }
}
```

# Memory Diagrams (2)



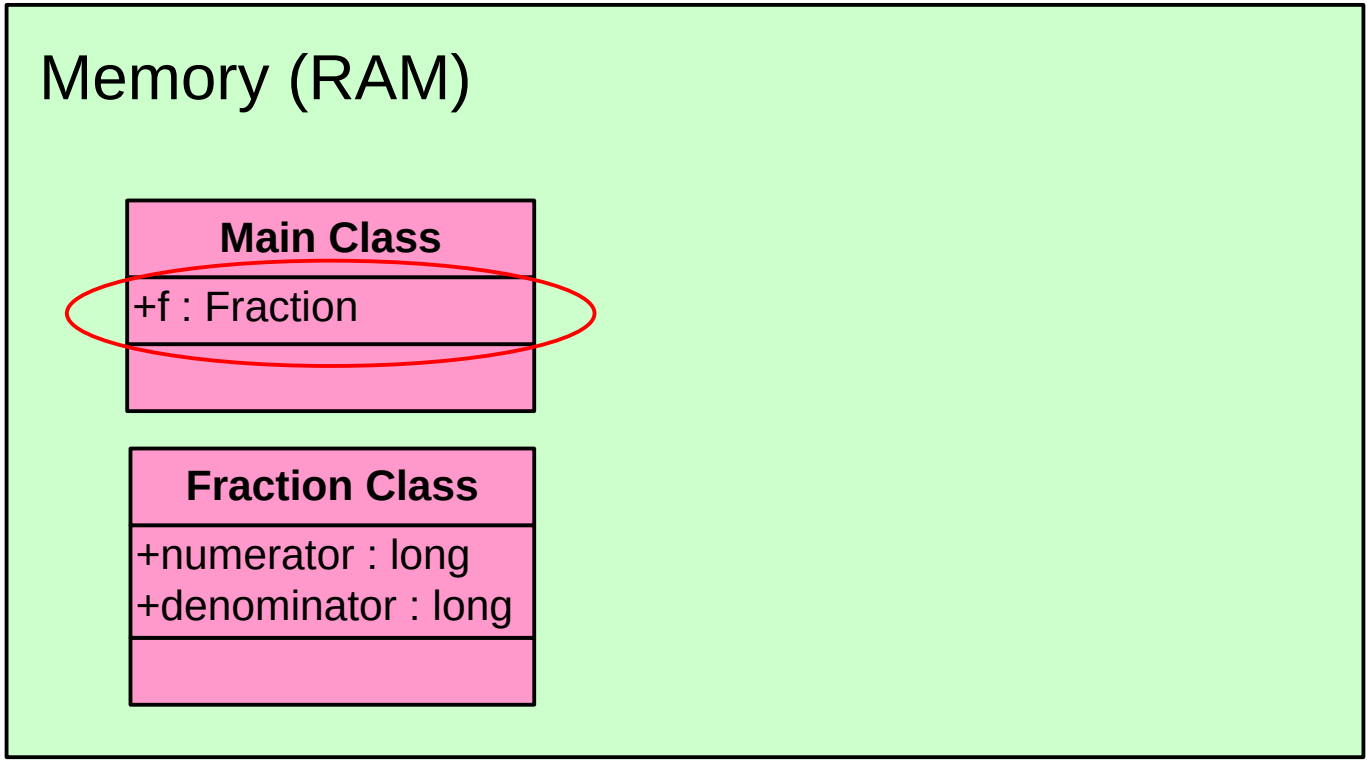
# Object Creation in Memory

1. Locate the class



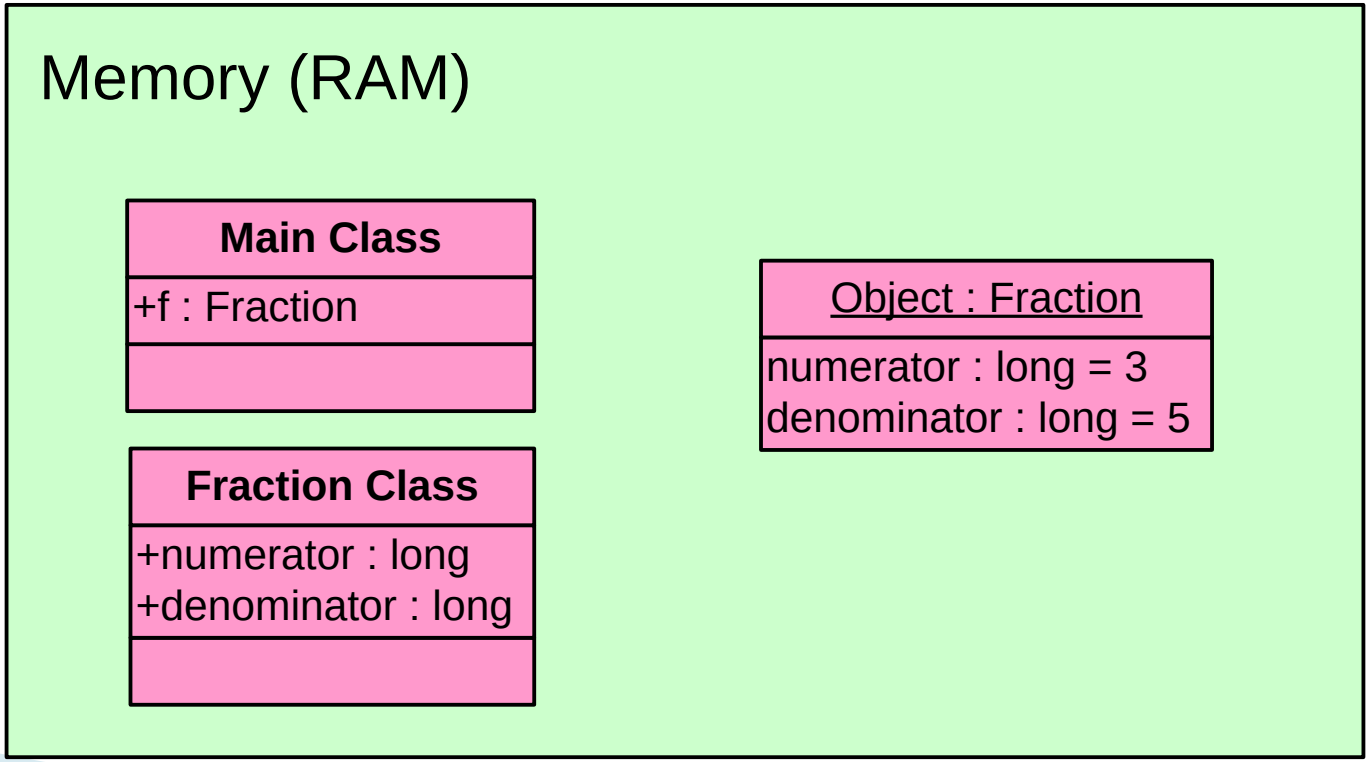
# Object Creation in Memory

2. Declare a reference



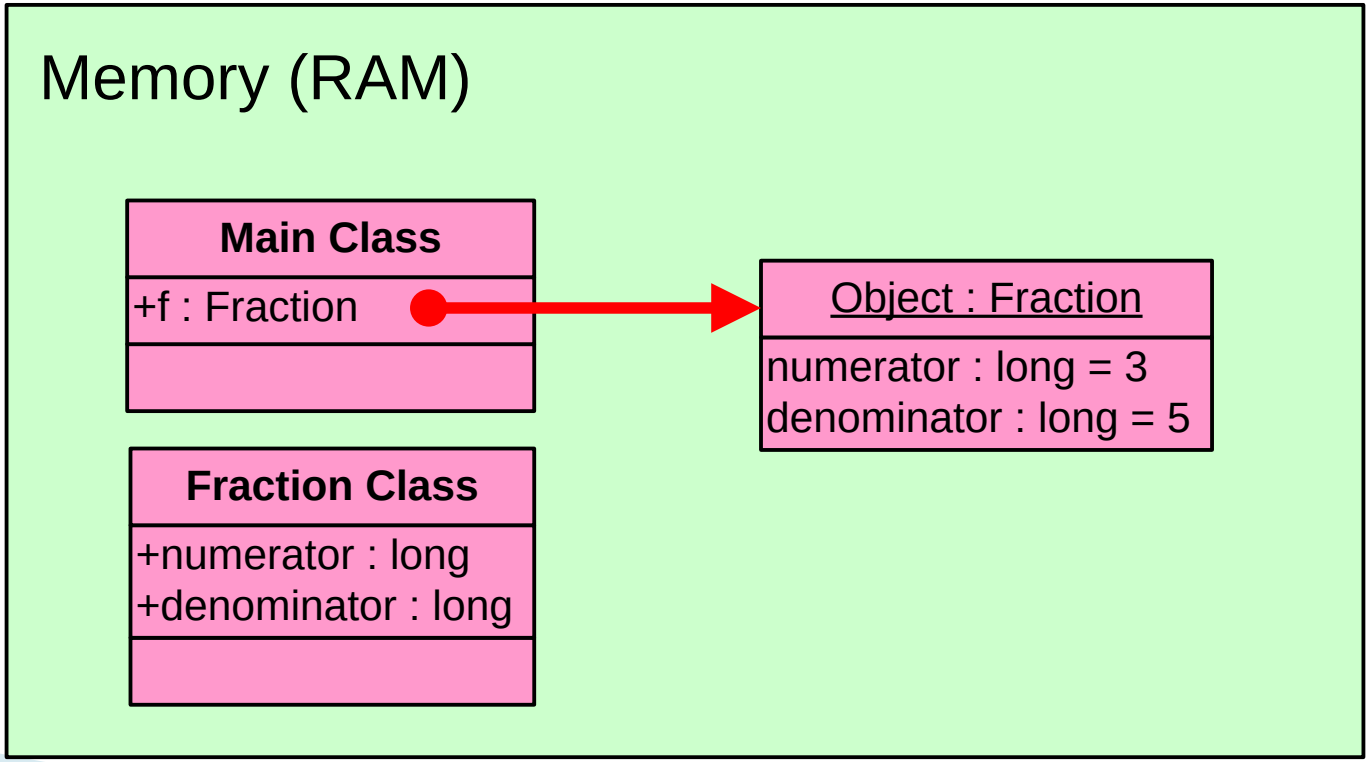
# Object Creation in Memory

3. Instantiate the class



# Object Creation in Memory

4. Assign a reference



# Using Objects (Example)

```
…
int width = 8;
int height = 5;
Rectangle3 r = new Rectangle3();
r.width = width;
r.height = height;
int rArea = r.getArea();
System.out.println(rArea);
…
```

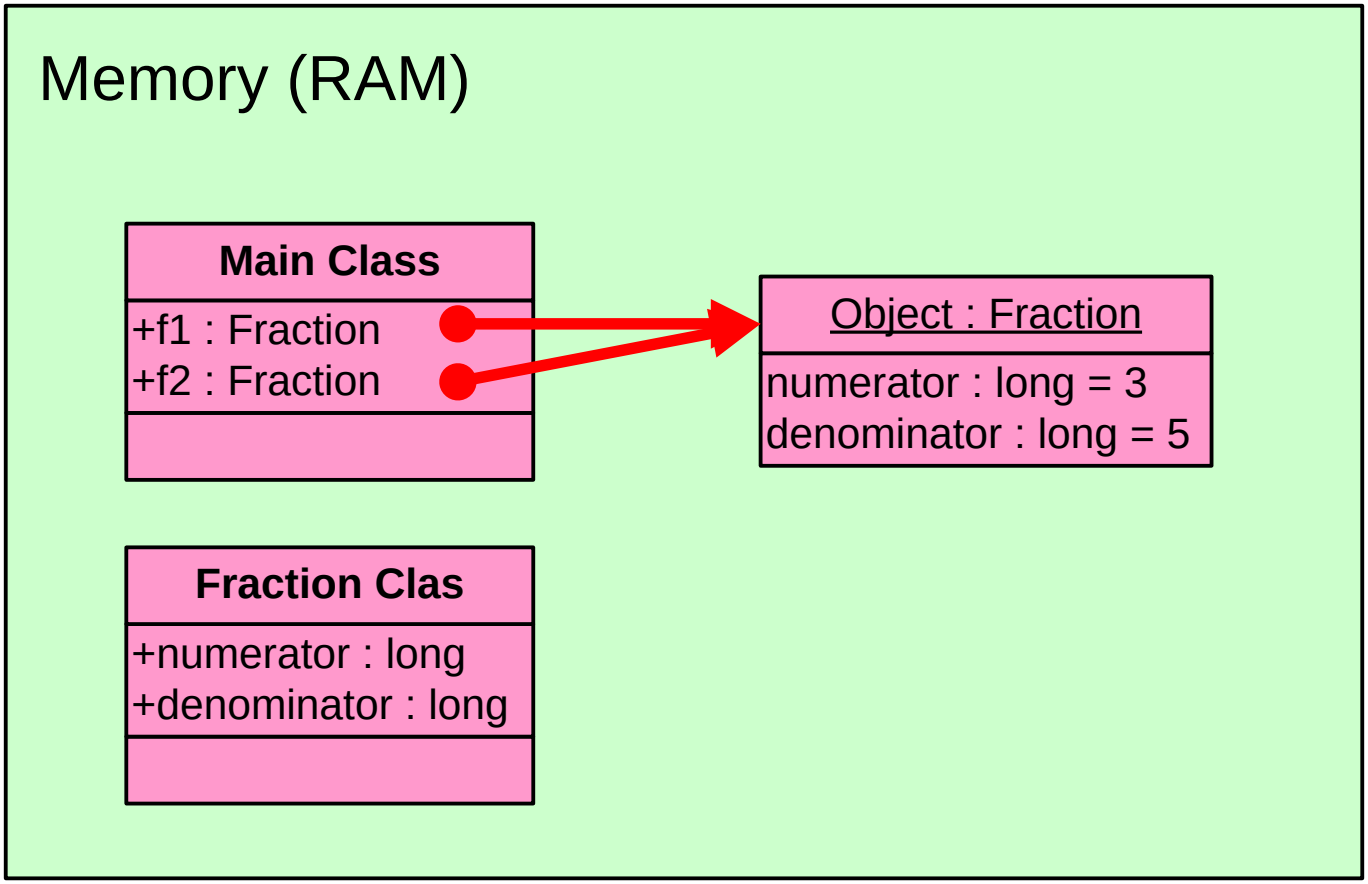
# Multiple References to an Object

- A reference can only point to one object at a time
- Multiple references can point to the same object
- Example

```
Fraction f1;
f1 = new Fraction(3, 5);
Fraction f2;
f2 = f1;   // both point to the same object
```

- State changes via one reference affects the object
- Object changes are visible via any reference to it

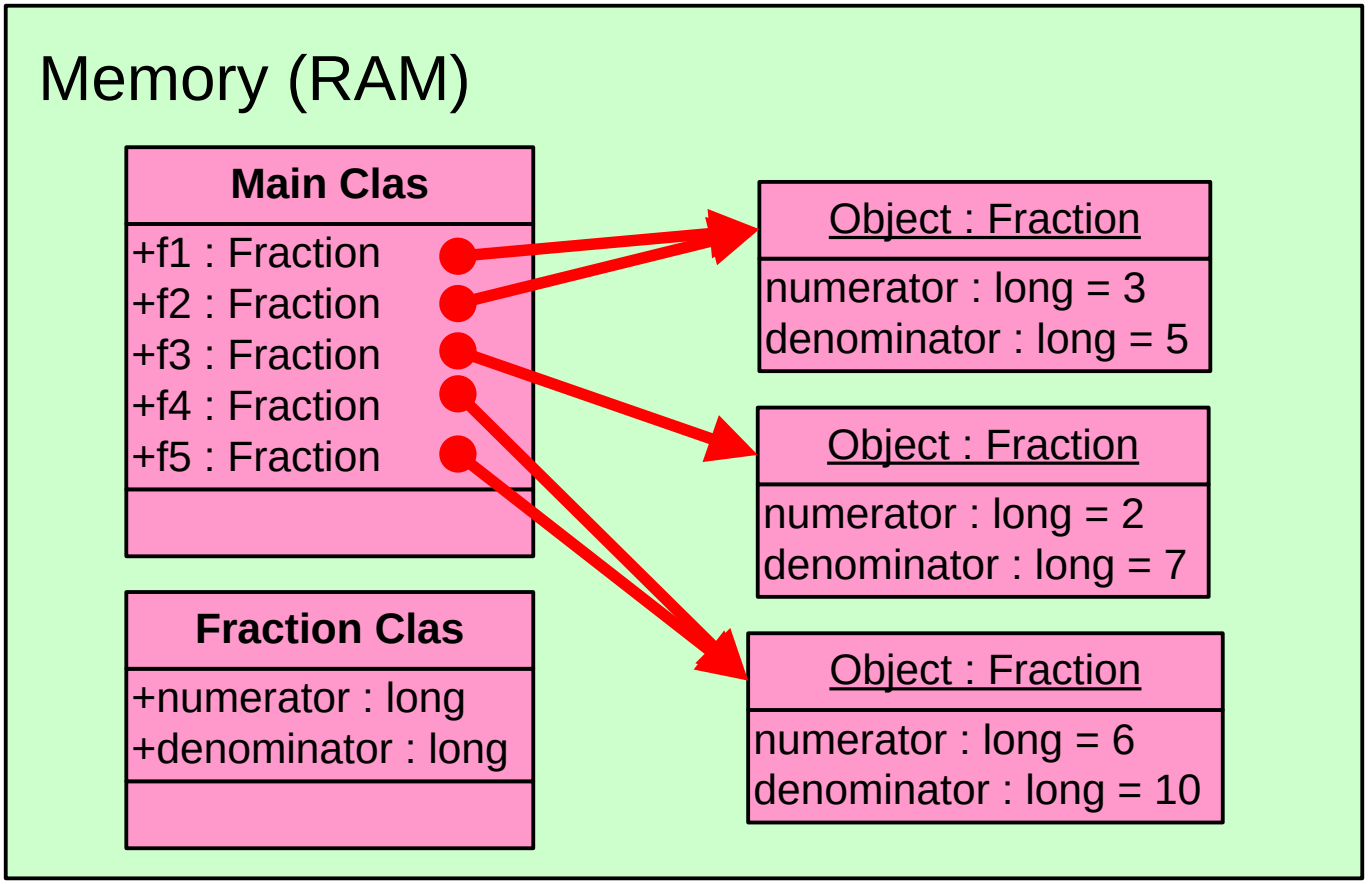
# Multiple References to an Object



# Object Equality

- Comparison using == operator only check memory address, not object state
- Comparison of object state requires use of the equals() method
- Example
  - *objRef1*.equals(*objRef2*);
- Definition of object equality defined by class implementer (in API)

# Object Equality



# Object Deletion (…sort of)

- In Java, the programmer cannot remove an object from memory
- Can orphan an object by removing referent to it
- Example

```
Fraction x = new Fraction(3, 5);
Fraction y = x;
y = new Fraction(4, 7);
x = null;
```

- Orphaned objects are cleared via garbage collection

# Development Process

- Task Analysis
- Algorithm Design
- Code Implementation
- Program Testing and Debugging

# Task Analysis

- Create a utility class and static method to convert kilometers to miles

- Input
  - km: the distance in kilometers
- Output
  - The distance in miles
- Throws
  - Exception if amount $< 0$

# Algorithm Design

- In this case, it involved a simple arithmetic operation:

$$\#miles = \#km \ / \ \#km\_per\_mile$$

# Code Implementation

```
/** This utility class converts metric to imperial. */
public class UnitConverter
{
	/** Converts the passed km value to miles.
	 * @param km distance in km
	 * @return distance in miles
	 * @throws Exception if value is negative */
	public static double kmToMiles(double km)
	{
		final double KM_PER_MI = 1.60934;
		if (km < 0)
		{
			throw new Exception("Negative value");
		}
		return km / KM_PER_MI;
	}
}
```

# Program Testing and Debugging

- Pick example inputs for the program
- Use a calculator to determine expected results
- Compare expected results with actual ones
- Identify any sources of error in the program
  - Debug using print statements to output variable values
- Correct any errors
- Retest program

- Covered in-depth in part 2