CSE 1710

Lectures 12

The Life of an Object

Background Material

- The examine the life of an object we make use of **memory diagrams**
  - readings from JBA concerning memory diagrams
    - sec 1.2.3
    - sec 3.3.1
    - sec 4.2.1, 4.2.2, 4.2.3

What are memory diagrams?

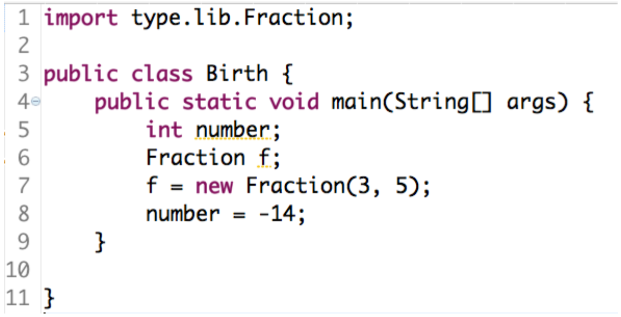
- a visualization of the heap space that is allocated to the java virtual machine (JVM) at run time
- the heap space is a portion of working memory used by the JVM for dynamic memory allocation
  - Key aspects of dynamic memory allocation
    - allocate memory to the Java program as the program needs it
    - free memory for re-use when it is no longer needed

JVM basics

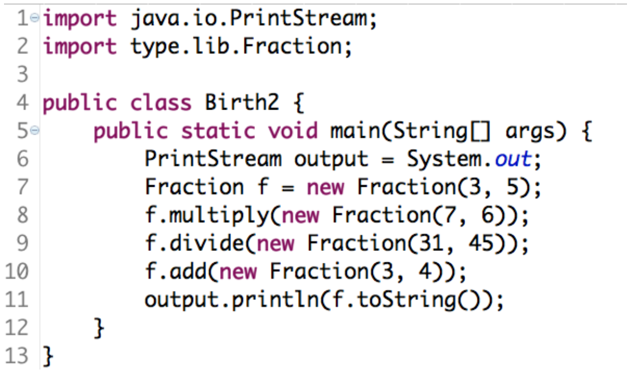
- when an app is compiled, the resulting *byte code* may depend on other ***class definitions***
  - the compiler checks the build path in order to locate these class definitions (**where** the *byte code* for those classes can be found on the file system)
  - early binding
- when an app is invoked, the JVM loads the *byte code* for class definitions when the services of that class are first required.
  - the JVM searches the class path for this byte code (looks for corresponding *.class files on the hard drive)
  - the JVM loads these class definitions into memory during run time

JVM basics

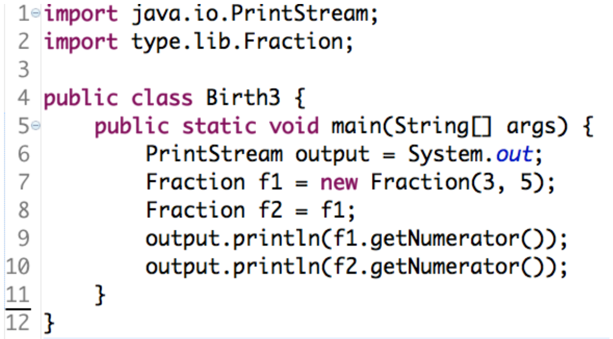
1. the class loader
   - loads the class definition that contains the main method
   - loads class definitions (byte code) of classes that are used by the app on demand
2. bytecode execution
   - execute the byte code that corresponds to the first statement of the main method.
   - then the byte code corresponding to the second line of the main method.
   - And so on...
   - until there are no further statements to be invoked.
3. Tidy shut down.



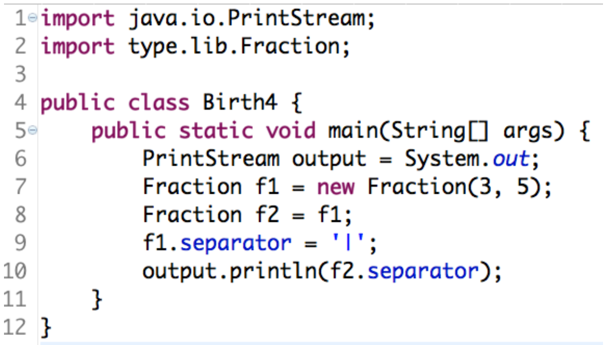
```java
import type.lib.Fraction;

public class Birth {
    public static void main(String[] args) {
        int number;
        Fraction f;
        f = new Fraction(3, 5);
        number = -14;
    }

}
```



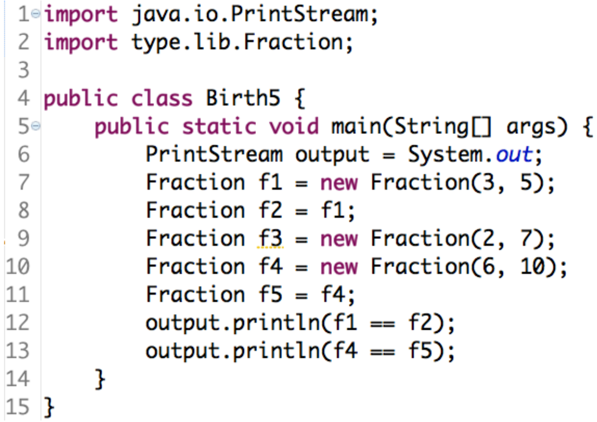
```java
import java.io.PrintStream;
import type.lib.Fraction;

public class Birth2 {
    public static void main(String[] args) {
        PrintStream output = System.out;
        Fraction f = new Fraction(3, 5);
        f.multiply(new Fraction(7, 6));
        f.divide(new Fraction(31, 45));
        f.add(new Fraction(3, 4));
        output.println(f.toString());
    }
}
```



```java
import java.io.PrintStream;
import type.lib.Fraction;

public class Birth3 {
    public static void main(String[] args) {
        PrintStream output = System.out;
        Fraction f1 = new Fraction(3, 5);
        Fraction f2 = f1;
        output.println(f1.getNumerator());
        output.println(f2.getNumerator());
    }
}
```



```java
import java.io.PrintStream;
import type.lib.Fraction;

public class Birth4 {
    public static void main(String[] args) {
        PrintStream output = System.out;
        Fraction f1 = new Fraction(3, 5);
        Fraction f2 = f1;
        f1.separator = '|';
        output.println(f2.separator);
    }
}
```



```java
import java.io.PrintStream;
import type.lib.Fraction;

public class Birth5 {
    public static void main(String[] args) {
        PrintStream output = System.out;
        Fraction f1 = new Fraction(3, 5);
        Fraction f2 = f1;
        Fraction f3 = new Fraction(2, 7);
        Fraction f4 = new Fraction(6, 10);
        Fraction f5 = f4;
        output.println(f1 == f2);
        output.println(f4 == f5);
    }
}
```