CSE 1710

Lectures 11, 12

Memory Diagrams, Input Validation

Background Material

- Selected 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 will contain the specification of which classes, if any, are required during run time.
  - the byte code for those classes, in turn, will indicate the class files upon which they depend
  - thus, an app may depend on many classes, either directly or indirectly
- when an app is invoked by the JVM, all of the classes upon which the app depends must be available to the JVM
  - the JVM must be able to find the corresponding *.class file on the hard drive.
  - the class path says where to look

JVM basics

- when an app is compiled, the resulting byte code will contain the specification of which classes, if any, are required during run time.
  - the byte code for those classes, in turn, will indicate the class files upon which they depend
  - thus, an app may depend on many classes, either directly or indirectly
- when an app is invoked by the JVM, all of the classes upon which the app depends must be available to the JVM
  - the JVM must be able to find the corresponding *.class file on the hard drive.
  - the class path says where to look

JVM basics

1. the class loader
   - loads the class definition that contains the main method
   - loads any and all class definitions (byte code) of classes that are used by the app*
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.

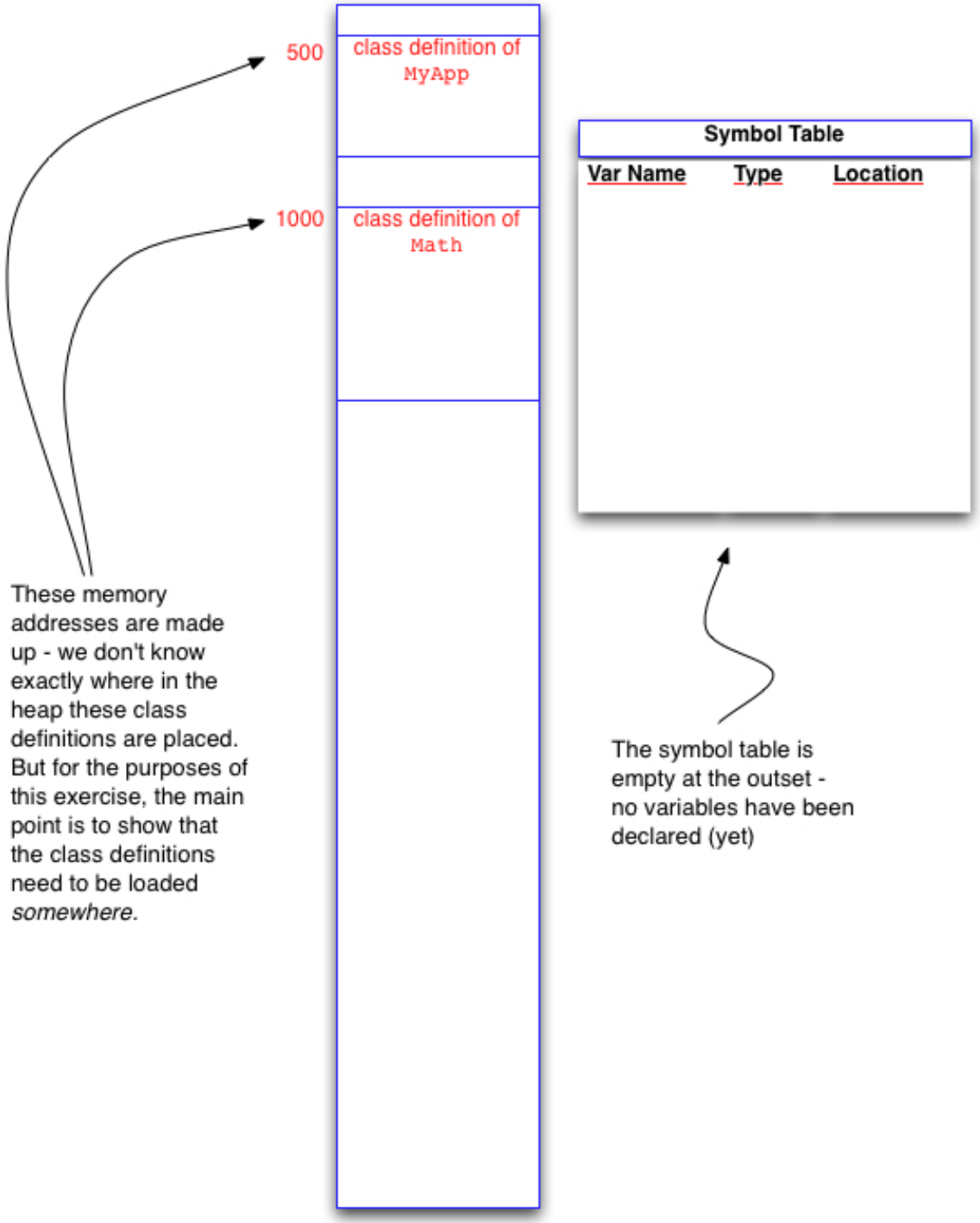
*strictly speaking, some classes are loaded on demand, but this complicates things and we will assume for the purposes of this course that all of the required classes are loaded at the outset).

Example

```
1  import java.lang.Math;
2
3  public class MyApp {
4
5    public static void main(String[] args) {
6      double val;
7      val = Math.PI;
8    }
9
10 }
```

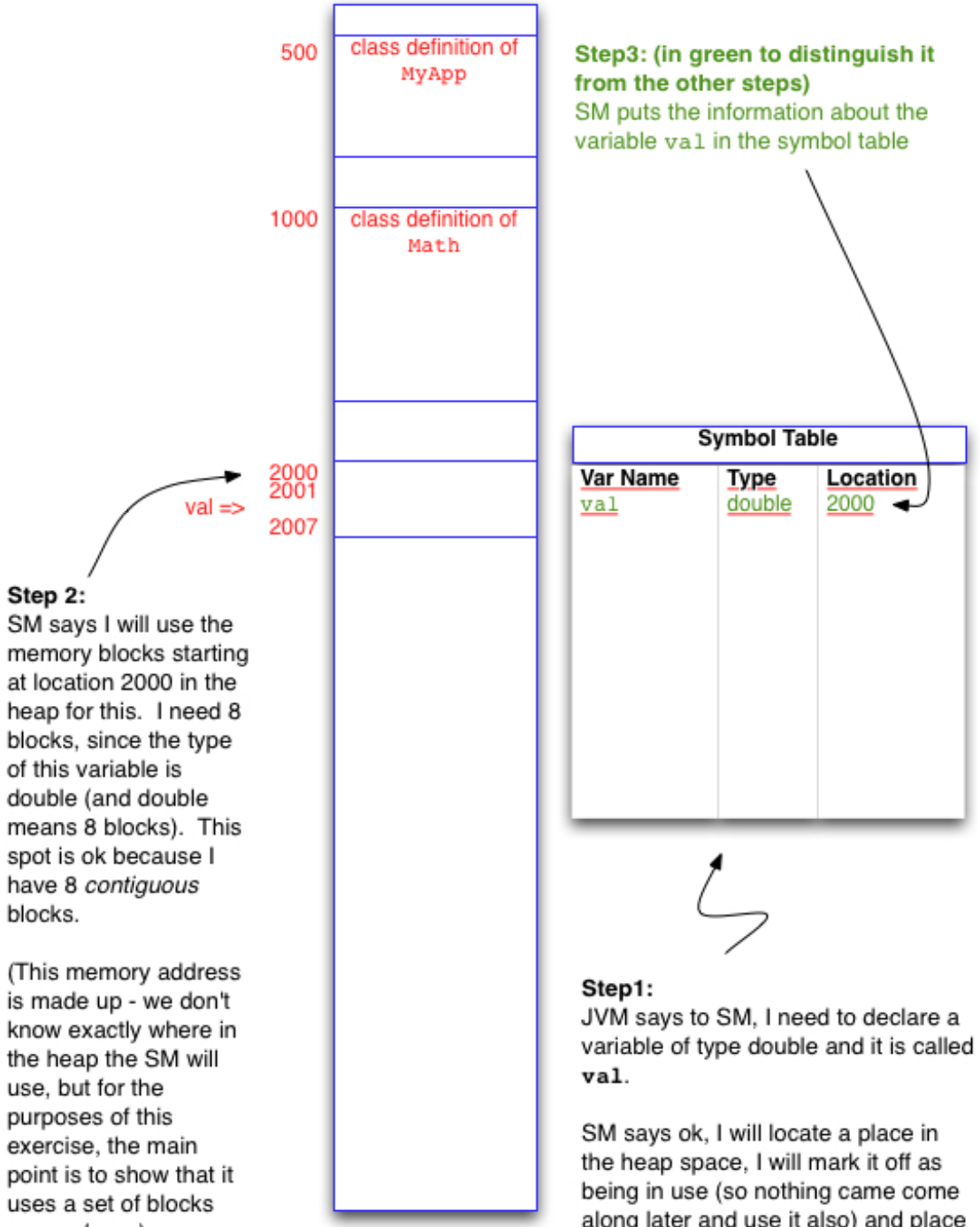
Memory Diagrams

up to but not including line 6



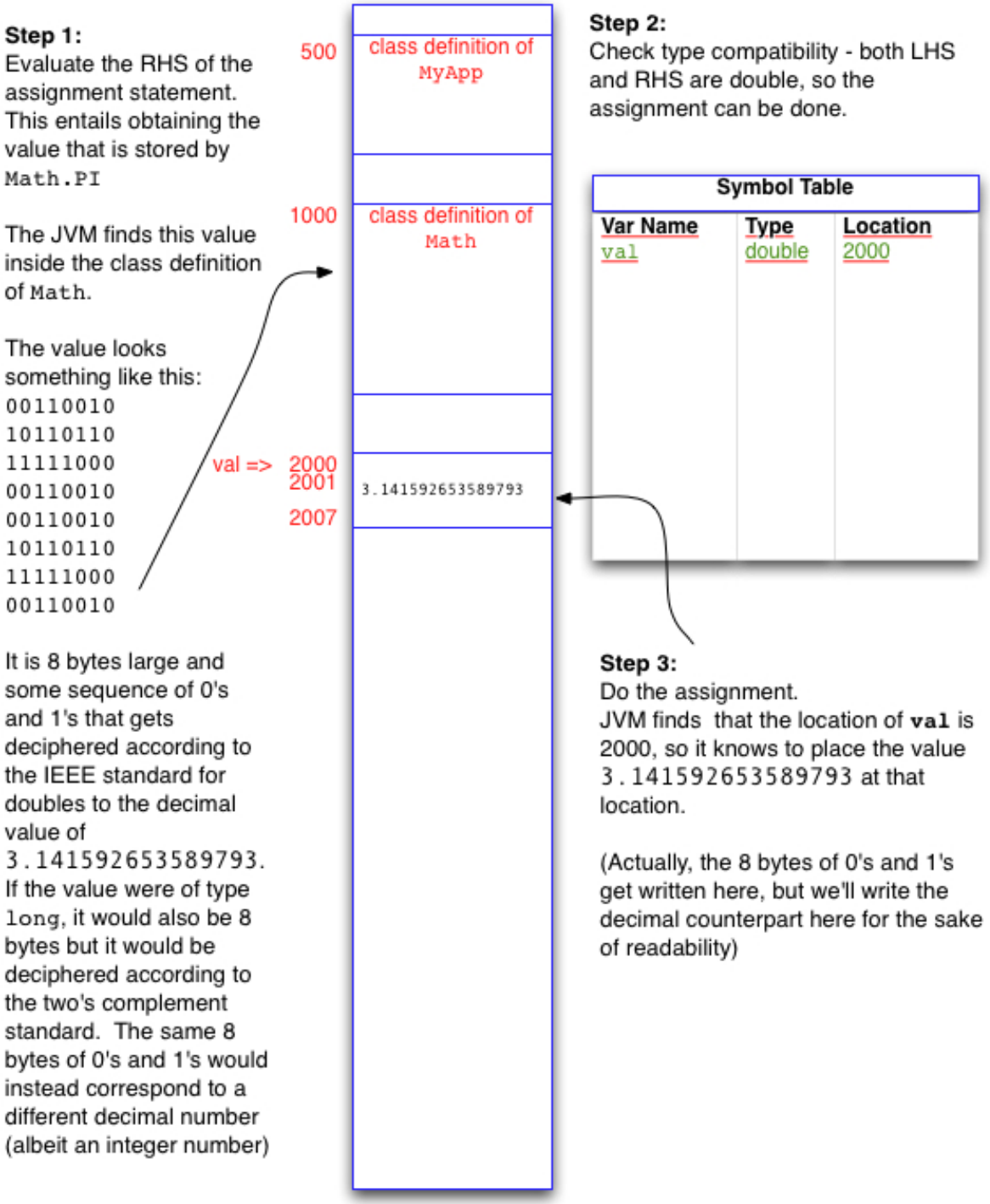
Memory Diagrams

up to and including line 6 (variable declaration)



Memory Diagrams

up to and including line 7 (variable assignment)

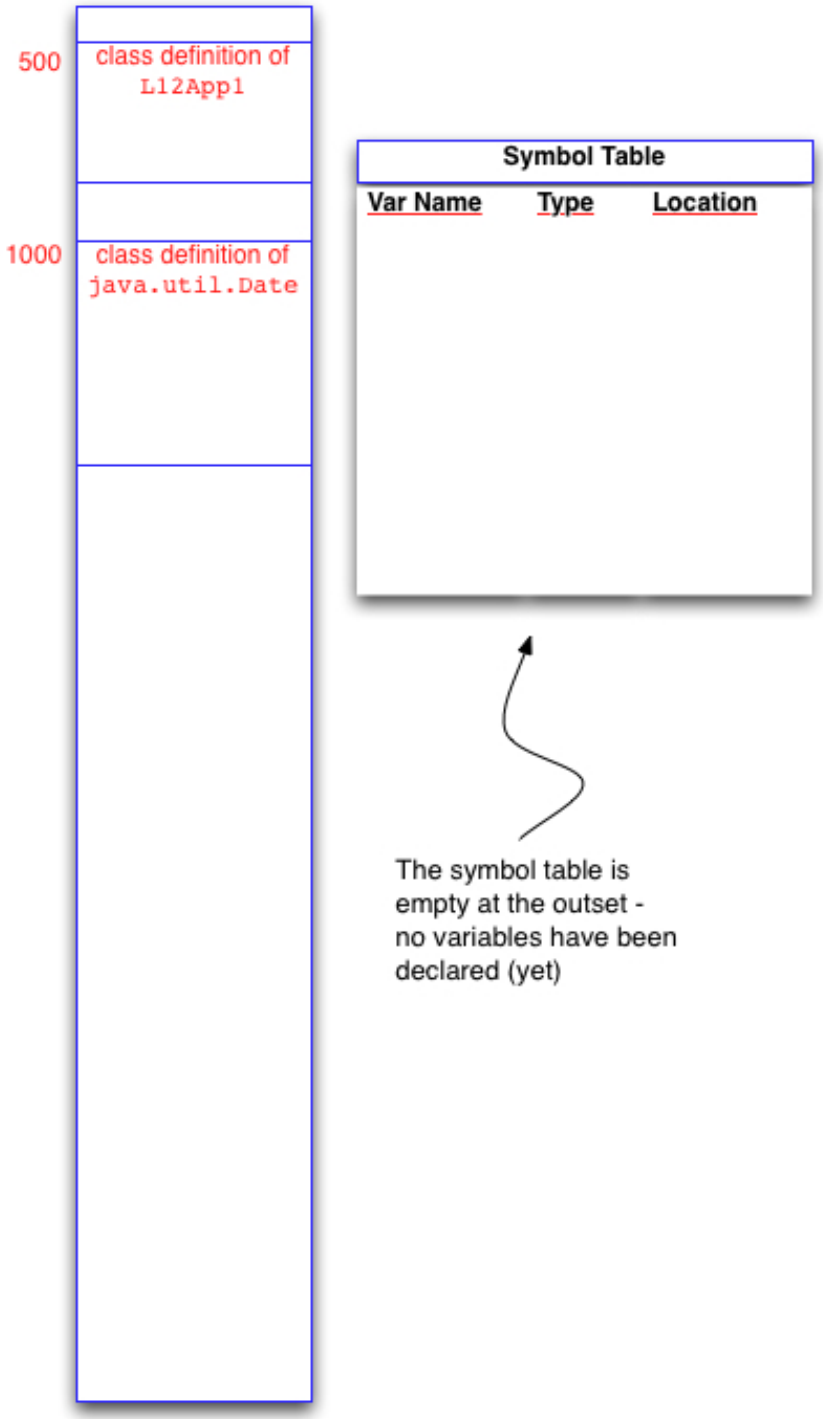


Example – object creation

```
1  import java.util.Date;
2
3  public class L12App1 {
4      public static void main(String[] args) {
5          Date d;
6          d = new Date();
7      }
8  }
```

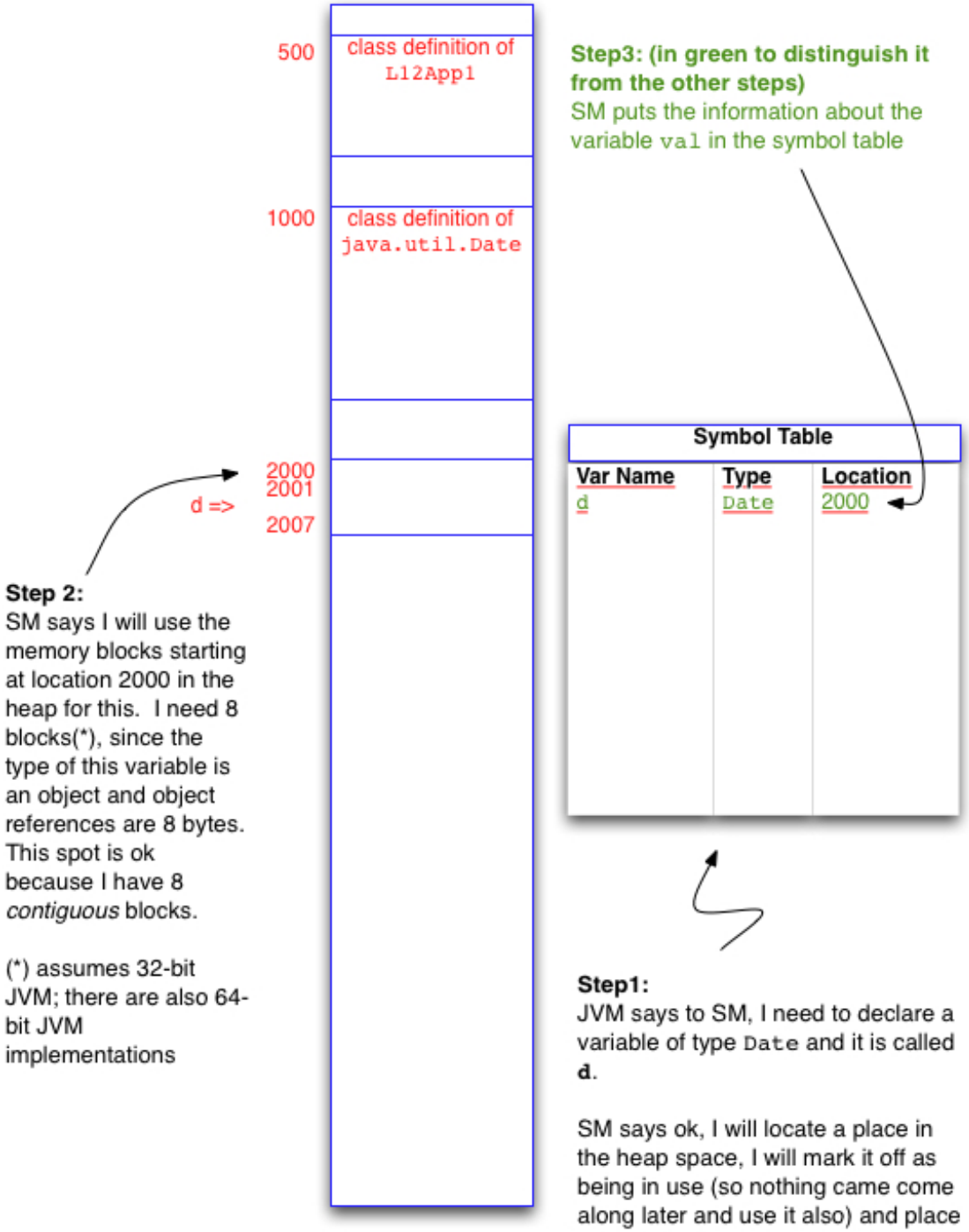
Memory Diagrams

up to but not
including line 5



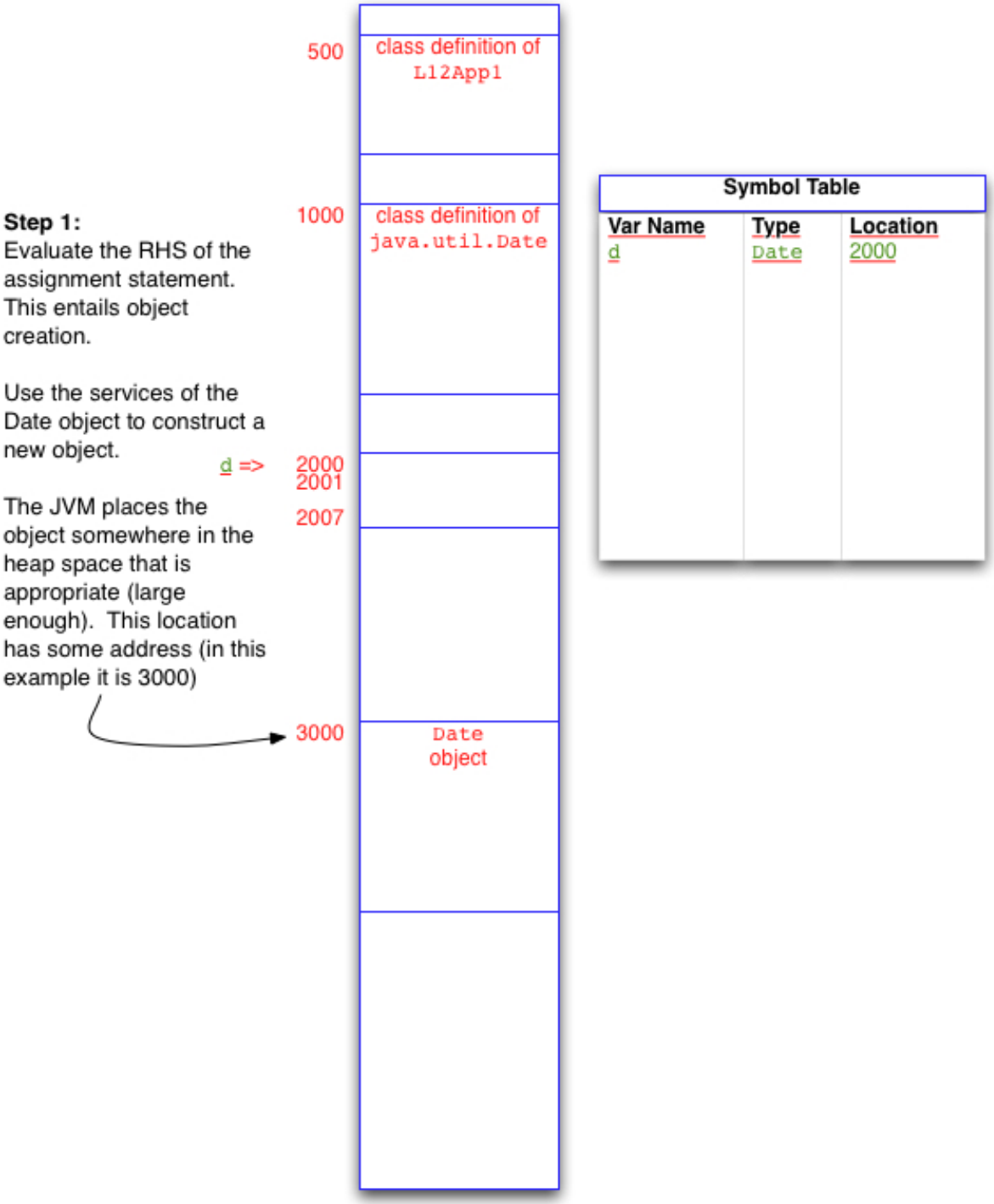
Memory Diagrams

up to and including line 5 (the variable declaration)



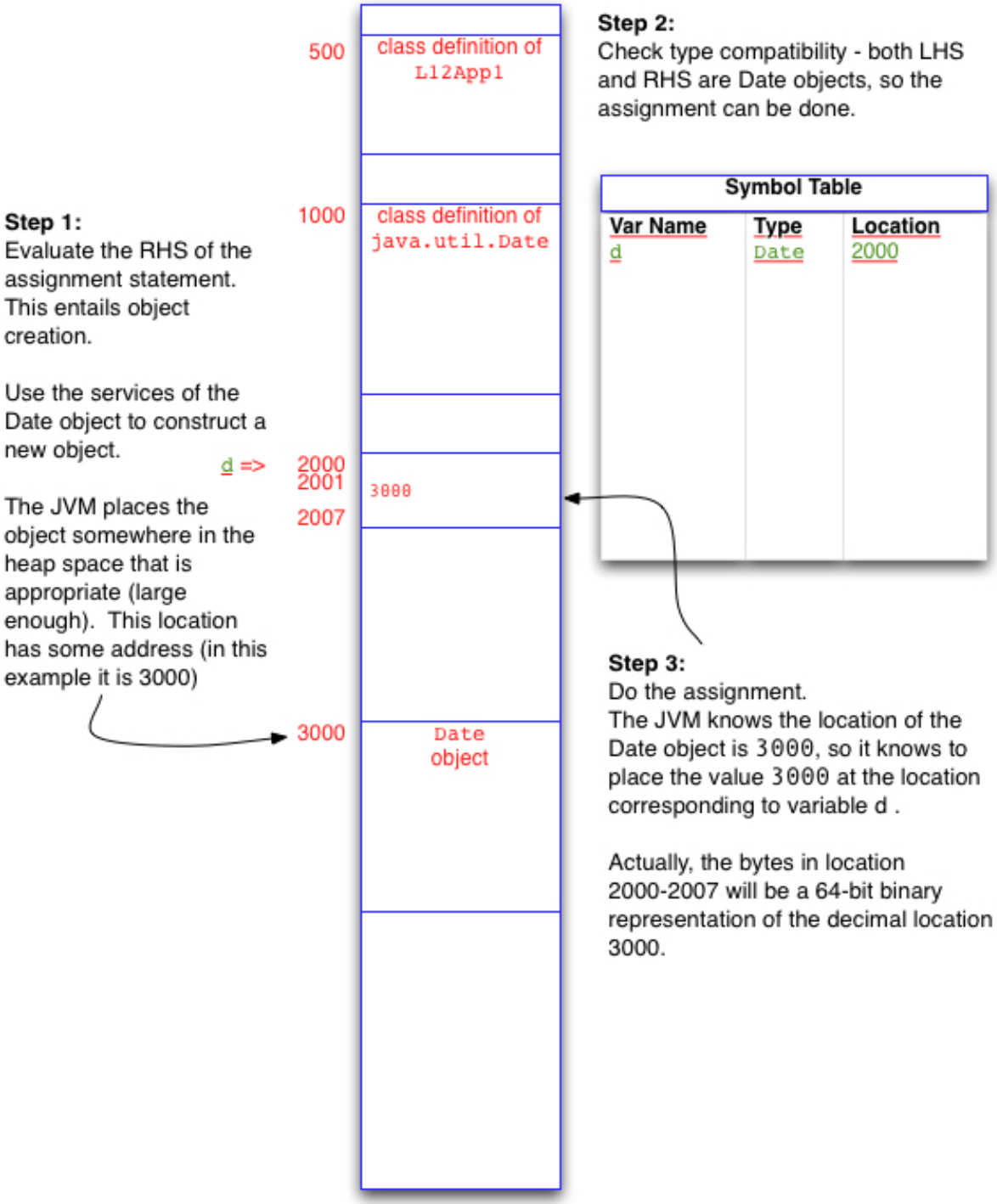
Memory Diagrams

up to and including the RHS of line 6 (object construction)



Memory Diagrams

up to and including the entire line 6 (object construction and reference assignment)



Example – object creation

```
1  import java.util.Date;
2
3  public class L12App2 {
4      public static void main(String[] args) {
5          Date d;
6          d = new Date();
7          Date d1;
8          d1 = new Date();
9          // is d==d1?
10     }
11 }
```

Example – object equality

```
1  import java.util.Date;
2
3  public class L12App3 {
4      public static void main(String[] args) {
5          long now = System.currentTimeMillis();
6          Date d;
7          d = new Date(now);
8          Date d1;
9          d1 = new Date(now);
10         // is d==d1?
11         System.out.println(d==d1);
12         System.out.println(d.equals(d1));
13         //how do you explain this???
14
15     }
16 }
```

Input Validation – Exception-Based Approach

```
boolean cond = amount < 0;
…
String msg = “The inputted amount was negative”;
…
ToolBox.crash(cond, msg);
```

Input Validation – Message-Based Approach

```
boolean cond = amount < 0;
…
String msg = “The inputted amount was negative”;
…
if (cond) {
     output.println(msg);
     }
else {
     //rest of program
}
```

Input Validation – Friendly Approach

```
boolean cond = amount < 0;
…
String msg = “The inputted amount was negative”;
…
for (n=input.nextInt(); n<=0; n=input.nextInt()
{
     output.println(msg);
}
```