Variables, Strings and Arrays in Java

Which variable is referenced?

```
public class B0 {
   public int i0 = 0;
   public Integer ii0 = new Integer(0);
   public B0( ) { }
}
```

i0, ii0 visible in B0

```
public class B1 extends B0 {
   public int i1 = 1;
   public Integer ii1 = new Integer(1);
   public B1( ) { }
}
```

i0, ii0 from B0, i1 and ii1 from B1 are all visible in B1

```
public class B2 extends B1 {
   public int i0 = 2;
   public int i1 = 2;
   public Integer ii0 = new Integer(2);
   public Integer ii1 = new Integer(2);
   public B2( ) { }
   public int getB1i1( ) {return super.i1;}
}
```

i0, ii0, i1 and ii1 from B2 are all visible in B2. getB1i1 returns the i1 declared in B1.

Referencing the variables

```
public class T {

   public static void main(String[ ] args) {

      B0 b0 = new B0( );
      B1 b1 = new B1( );
      B2 b2 = new B2( );

      System.out.println("b0.i0 = "+b0.i0); // b0.i0 = 0
      // NOT DECLARED System.out.println("b0.i1 = "+b0.i1);
      System.out.println("b0.ii0 = "+b0.ii0.intValue( )); // b0.ii0 = 0
      // NOT DECLARED System.out.println("b0.ii1 = "+b0.ii1.intValue( ));

      System.out.println("b1.i0 = "+b1.i0); // b1.i0 = 0
      System.out.println("b1.i1 = "+b1.i1); // b1.i1 = 1
      System.out.println("b1.ii0 = "+b1.ii0.intValue( )); // b1.ii0 = 0
      System.out.println("b1.ii1 = "+b1.ii1.intValue( )); // b1.ii1 = 1
      . . .
   }
}
```

```
public class T {

   public static void main(String[ ] args) {
      . . .
      System.out.println("b2.i0 = "+b2.i0); // ???
      System.out.println("b2.i1 = "+b2.i1); // b2.i1 = 2
      System.out.println("b2.ii0 = "+b2.ii0.intValue( )); // ???
      System.out.println("b2.ii1 = "+b2.ii1.intValue( )); // b2.ii1 = 2
      System.out.println("b2.super i1  = "+b2.getB1i1( )); // ???

      b1 = b2;
      System.out.println("b1.i0 = "+b1.i0); // ????
      System.out.println("b1.i1 = "+b1.i1); // ???
      System.out.println("b1.ii0 = "+b1.ii0.intValue( )); // ???
      System.out.println("b1.ii1 = "+b1.ii1.intValue( )); // ???

   }
}
```

```
public class T {

   public static void main(String[] args) {
      . . .
      System.out.println("b2.i0 = "+b2.i0); // b2.i0 = 2
      System.out.println("b2.i1 = "+b2.i1); // b2.i1 = 2
      System.out.println("b2.ii0 = "+b2.ii0.intValue( )); // b2.ii0 = 2
      System.out.println("b2.ii1 = "+b2.ii1.intValue( )); // b2.ii1 = 2
      System.out.println("b2.super i1  = "+b2.getB1i1( )); // b2.super i1  = 1

      b1 = b2;
      System.out.println("b1.i0 = "+b1.i0); // b1.i0 = 0
      System.out.println("b1.i1 = "+b1.i1); // b1.i1 = 1
      System.out.println("b1.ii0 = "+b1.ii0.intValue( )); // b1.ii0 = 0
      System.out.println("b1.ii1 = "+b1.ii1.intValue( )); // b1.ii1 = 1
      What if we tried to call b1.getB1i1( ) here?
   }
}
```

Java has built-in strings

- String and StringBuffer Classes
- String objects are *immutable*
  - Cannot be changed after creation
  - Can be garbage collected by the Java system if there are no references to it.
    - Garbage Collection, or GC, is kind of analogous to having the system do a C *free*, automatically.
- String literals are double quoted, i.e. “this is a string” is a string literal with the value of *this is a string*
- A given string literal is **usually** only stored once in memory

String uniqueness example

```
//StringLiteralUniqueness.java
class X { public static String strX = "hello"; }                    //(A)
class Y { public static String strY = "hello"; }                    //(B)
class Z { public static String strZ = "hell" + "o"; }               //(C)

class Test {
    public static void main( String[] args ) {
        // output: true
        System.out.println( X.strX == Y.strY );                     //(D)

        // output: true
        System.out.println( X.strX == Z.strZ );                     //(E)

        String s1 = "hel";
        String s2 = "lo";

        // output: false
        System.out.println( X.strX == ( s1 + s2 ) );                //(F)

        // output: true
        System.out.println( X.strX == (s1 + s2).intern() );         //(G)
    }
}
```

String uniqueness example

```
//StringLiteralUniqueness.java
class X { public static String strX = "hello"; }
class Y { public static String strY = "hello"; }
class Z { public static String strZ = "hell" + "o"; }

class Test {
    public static void main( String[] args ) {
        // output: true
        System.out.println( X.strX == Y.strY );
        // output: true
        System.out.println( X.strX == Z.strZ );
        String s1 = "hel";
        String s2 = "lo";
        // output: false
        System.out.println( X.strX == ( s1 + s2 ) );
        // output: true
        System.out.println( X.strX == (s1 + s2).intern() );
    }
}
```

String uniqueness example

class X { public static String strX = "hello"; }
class Y { public static String strY = "hello"; }
class Z { public static String strZ = "hell" + "o"; }

```
class Test {
    public static void main( String[] args ) {
        // output: true
        System.out.println( X.strX == Y.strY );

        // output: true
        System.out.println( X.strX == Z.strZ );

        String s1 = "hel";
        String s2 = "lo";

        // output: false
        System.out.println( X.strX == ( s1 + s2 ) );             //(F)

        // output: true
        System.out.println( X.strX == (s1 + s2).intern() );      //(G)
    }
}
```

String uniqueness example

```
class X { public static String strX = "hello"; }
class Y { public static String strY = "hello"; }
class Z { public static String strZ = "hell" + "o"; }

class Test {
   public static void main( String[] args ) {
      // output: true
      System.out.println( X.strX == Y.strY );

      // output: true
      System.out.println( X.strX == Z.strZ );

      String s1 = "hel";
      String s2 = "lo";
      // output: false first, then true
      System.out.println( X.strX == ( s1 + s2 ) );
      System.out.println( X.strX == (s1 + s2).intern() );
   }
}
```

How strings should be compared

```
class X { public static String strX = "hello"; }
class Y { public static String strY = "hello"; }
class Z { public static String strZ = "hell" + "o"; }

class Test {
   public static void main( String[] args ) {
      // output: true
      System.out.println( X.strX == Y.strY );

      // output: true
      System.out.println( X.strX == Z.strZ );

      String s1 = "hel";
      String s2 = "lo";
      // output: false first, then true
      System.out.println( X.strX.equals(s1 + s2));
      System.out.println( X.strX == (s1 + s2).intern() );
   }
}
```

Constructing Strings and StringBuffers

- Strings are usually straightforward
  - String str = new String(“hello there”);
  - String str = “hello there”;
  - String str = new String( ); // empty
  - String str = “”;
- A String **is not** a StringBuffer
  - StringBuffer sb = str; // is wrong!
  - StringBuffer sb = “hello world” // is wrong!

Constructing StringBuffers

- Empty StringBuffer
  - StringBuffer sb = new StringBuffer(“”);
  - StringBuffer sb = new StringBuffer( );
- Non-empty StringBuffer
  - like String, except use StringBuffer
    - storage occupied is length of argument (in characters) + something
    - *length* is the number of characters
  - StringBuffer sb = new StringBuffer(1024);

Other String Operations

- Constructors exist to create Strings from integers, Arrays of char, etc.
- Can do insert, substitute, access individual characters, and many other things possible with, e.g., Python

Remember Strings are immutable

```
public class Immut {

   public static void main(String[] args) {
      String s1 = new String("012345");
      String s2 = s1;
      s1 = s1.replace('2', 'R');
      System.out.println("s1 = "+s1+", s2 = "+s2);
   }
}
```

```
$ java Immut
s1 = 01R345, s2 = 012345
```

StringBuffers are mutable

```
public class Mut {

   public static void main(String[] args) {
      StringBuffer sb1 = new StringBuffer("012345");
      StringBuffer sb2 = sb1;
      sb1 = sb1.replace(2, 2, "R");
      System.out.println("sb1 = "+sb1+", sb2 = "+sb2);
   }
}
```

```
$ java Mut
sb1 = 01R2345, sb2 = 01R2345
```

In General . . .

- Use StringBuffers
  - when you want mutability
  - Mutability desirable when doing I/O, editing strings, etc.
- Use String
  - When you don’t want mutability
  - Saves some storage
  - You don’t need to worry about who else is referencing the changed String

Java Arrays

Java array declaration

```
// 3 elements with in indices 0, 1 and 2
int [ ] data = new int[3];
String [ ] strs = new String[3]; // Each element holds a reference
        // to s string object. There is no way to put an object
        // itself in an array

// an array with initialization
int [ ] data2 = new int[ ] {1, 2, 3};
String [ ] strs2 = new String[ ] {“s1”, “s2”, “s3”};
// an illegal declaration
int [ ] data3 = new int[3] {1, 2, 3};
```

Arrays are objects

Given: class User {String name; int age;}

An array
User[ ] userList = new User[10];

can be declared.
User[ ] is a new user defined type.

Can say: Object o = userList; User[ ] u2 = (User[ ]) o;