04-0: **Types in Java**

- Primative Types
  - Hold simple values
  - Can be stored on the stack
    - (but can be stored on the heap if they are instance variables in classes)
  - integers: **byte** (8 bits), **short** (16 bit) **int** (32 bit) **long** (64 bit)
  - real numbers: **float** (32 bit) **double** (64 bit)
  - **boolean**: true or false value
  - **char**: single character (16 bit, unicode)
    - in C, a char is 8 bits, uses ASCII

04-1: **Types in Java**

- Objects
  - Collection of data and methods
  - Always stored on the heap
    - Pointer to object can be on the stack
  - Created with a call to “new”

04-2: **Strings**

- Strings in Java are objects
- Contain both methods and data
  - Data is the sequence of characters (type **char**) that make up the string
  - Strings have a whole bunch of methods for string processing

04-3: **Strings**

- Strings in Java are objects
  - Strings are stored on the heap, like all other objects
  - Data is stored as an array of characters (more on arrays next week. Similar to python lists)

04-4: **String Literals**

```
String s;
s = "Dog";
```

- ”Dog” is called a *String Literal*
  - Anything in quotation marks is a string literal
  - `System.out.println("Hello There")`

04-5: **String Literals**

- Any time there is a string literal in your code, there is an implicit call to “new”

- A new string object is created on the heap
- Data is filled in to match the characters in the string literal
- Pointer to that object is returned

```
String s;
s = "MyString"; // Implicit call to new here!
```

04-6: **Stack vs. Heap I**

```
public void foo()
{
    int x = 99;
    char y = 'c';
    String z = "c";
    String r = "cat";
    float w = 3.14;
}
```

04-7: **Immutable Strings**

- Strings are *immutable*
- Once you create a string, you can't change it.

```
String s = "Car";   // Create a block of memory containing 'car'
                    // Return a pointer to this block of memory

unknown.foo(s);     // This function can't mess with contents of s

System.out.println(S);   // s is guaranteed to be "Car" here
```

04-8: **Immutable Strings**

- String *objects* are immutable
  - Once a string object is created, it can't be changed
- String *variables* can be changed
  - Create a new String object, assign it to the variable

```
String s = "dog";
s = "cat";
```

04-9: **"Mutable" Objects**

```
public class ICanChange
{
    private int x;

    public ICanChange(int initialX)
    {
        this.x = initialX;
    }
    public int getX()
    {
        return this.x;
    }
    publc void setX(int newX)
    {
        this.x = newX;
    }
}
```

04-10: **"Mutable" Objects**

```
ICanChange c = new ICanChange(4);
c.setX(11);  // Changed the value in object
             //  c points to
System.out.println(c.getX());
```

- Created an object of type ICanChange
- Changed the data within that object

04-11: **“Mutable” Objects**

```
ICanChange c = new ICanChange(4);
c = new ICanChange(11);
System.out.println(c.getX());
```

- Created an object of type ICanChange, with value 4
- Created a *new* object of type ICanChange, with value 11
    - Throw away the old object

04-12: **“Mutable” Objects**

```
ICanChange c = new ICanChange(4);
StrangeClass s = new StrangeClass(); // Don’t know what this does ...

s.foo(c);

System.out.println(c.getX());
```

04-13: **“Mutable” Objects**

```
public class StrangeClass
{
    void foo(ICanChange a)
    {
        a.setX(99);
    }
}
```

04-14: **“Immutable” Object**

```
public class ICantChange
{
    private int x;

    public ICanChange(int initialX)
    {
        this.x = initialX;
    }
    public int getX()
    {
        return this.x;
    }
}
```

04-15: **“Immutable” Object**

```
ICantChange c = new ICantChange(13);
System.out.println(c.getX());
c = new ICantChange(37);
System.out.println(c.getX());
```

- Create a new object, have c point to this new object
- Old object didn’t change, but the value of c did ....

04-16: **“Immutable” Object**

```
ICantChange c = new ICantChange(13);
Strange s = new Strange();

s.foo(c);
System.out.println(c.getX());
```

- Do we know anything about what the println will ouput?

04-17: **“Immutable” Objects**

```
public class Strange
{
    void foo(ICantChange icc)
    {
       // We can't change the value of x stored in icc
       //  directly (private, no setters)
       //
       //  Best we can do is change what icc points to ...
       icc = new ICantChange(99);
       //   icc.getX() would return 99 here, but what about
       //    the calling function?
    }
}
```

04-18: **Back to Strings**

- Strings are objects, like any other object
- Stored on the heap, but immutable
- Whole host of useful methods for string manipulation

04-19: **String Methods**

- public char charAt(int i): returns the character at index i (starting at 0)

```
String s = "cartwheel";
char c = s.charAt(2);
```

- What value would c now have?

04-20: **String Methods**

- public int length(): returns the length of the string

```
String s = "cartwheel";
int len = s.length();
```

- What value would len now have?

04-21: **String Methods**

- pubic String substring(int beginIndex): returns a new string, starting with beginIndex

```
String s = "cartwheel";
String s2 = s.substring(4);
```

- s2 would have the value “wheel”
- What value would s now have?

04-22: **String Methods**

- public String substring(int beginIndex, endIndex): returns a new string, starting with beginIndex, with last char at positn (endIndex - 1)

```
String s = "cartwheel;
String s2 = s.substring(1, 4);
```

- s2 would have the value “art”
- What value would s now have?

04-23: **String Methods**

- public String concat(String str) : returns a new string, consisting of this string concatenated with str

```
String s1 = "dog";
String s2 = "house";
Sring s3 = s1.concat(s2);
```

- s3 would have the value “doghouse”
- What value would s1, s2 have?

04-24: **String Methods**

- public String toLowerCase() : returns a new string, consisting of this string force into lower case

```
String s1 = "ThisIsAString";
String s2 = s1.toLowerCase();
```

- s2 would have the value “thisisastring”
- What value would s1 have?

04-25: **String Methods**

- public String toLowerCase() : returns a new string, consisting of this string force into lower case

```
String s1 = "ThisIsAString";
s1.toLowerCase();
```

- What value would s1 have?
- What just happened?