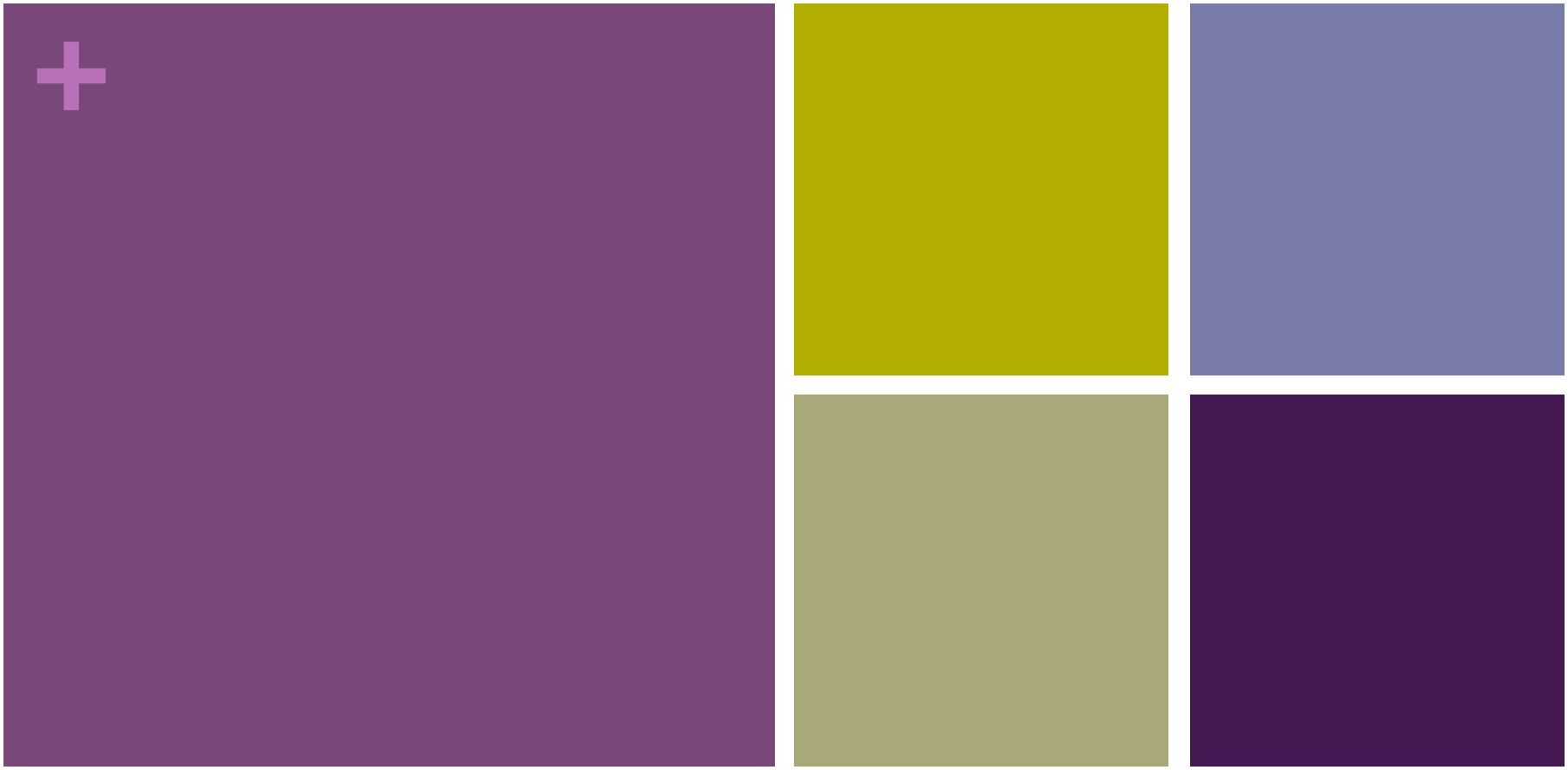


Java Variable Passing

Scope Copy Reference

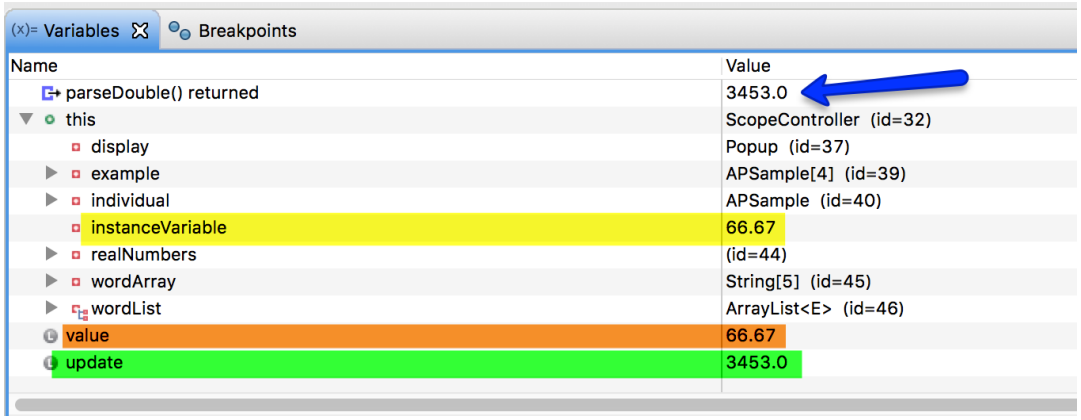
Cody Henrichsen

Primitive as Parameter

- ALWAYS A COPY!!!!!!
- Remember, primitives are really cheap
- Variable's value only changes if assigned to via return statement

Primitive Example

```
private void lookAtPrimitiveParameter(double value)
{
    double update = Double.parseDouble(display.getResponse("What is the new number"));
    value = update;
    display.displayMessage("Here is the parameter: "
                + value + ". Here is the local variable: " + update
                + ". Here is the instance variable: " + instanceVariable);
}
```



Object as Parameter

- Setters/Mutators will change the actual object
  - If you see the dot operator, change is happening!
- Assignment operator makes a new local only change
  - Just a temporary local variable that disappears when the squiggle ends
  - = ⇒ temp
- Remember that String is special. None of the methods in String change the objects value since Strings are immutable, only if you assign back into the original variable will the change made impact it at all.

Object Example

```
private void lookAtObjectParameter(APSample example)
{
    display.displayMessage(example.toString());
    display.displayMessage(example.hashCode() + ": is the parameter hash" + "\n"
                    +  individual.hashCode() + ": is the data member hash");
    example.setValue("look internal change to state stays");
    example = null;
    example = new APSample();
    display.displayMessage(example.hashCode() + ": is the updated parameter hash"
                    + "\n" + individual.hashCode() + ": is the data member hash");
    display.displayMessage(example.toString());

}
```

String Example

```
private String lookAtStringParameterWithReturn(String word)
{
    display.displayMessage("Remember; a Java String is immutable");

    word.substring(word.length() / 2);

    word = "Replaced the contents of the local variable only";

    return word;
}
```

Structure as Parameter

See object 😺

- Array is Hipster so no methods
  - `arrayName [index] = someValue;`
    - means you changed the value in the actual array
  - `arrayName = new Type [someInt];`
  - `arrayName = null;`
    - A BRAND New local array variable 💯
- List (et al)
  - `myList.set(n, someValue);`
  - `myList.add(myThing);`
  - `yourList.remove(0);`
  - `Other list methods…`
    - OK for the actual structure
    - Size and contents get modified
  - `myList = new ArrayList<Type>();`
    - 👀 the assignment operator so a new local variable

Structure example

```
private void lookAtStructureParameter(String [] words)
{
    display.displayMessage("The parameter is: " + words.length + " entries in length");
    words[4] = "asda";
    words = new String [3];
    words[0] = "temporary";
    words[1] = "values";
    words[2] = "don't last";
    display.displayMessage("The parameter is: " + words.length + " entries in length");
    wordArray[4] = wordArray[4] + ", but data can be modified";
}
```

For Each

- Only a "copy" of the value in the structure
  - The Assignment operator is USELESS!!!!!!!
    - Eclipse will warn you, other IDE's may not (👀DRJava)
      - Yellow line of uselessness
  - Setter/Mutator methods **DO** change values
  - You may have seen this in the AP Photo Lab via the zeroBlue method in the sample code
    - This lab is great to review this weekend BTW

For Each Example

NO change

```
private void showForEachDoesntChangeTheVariable()
{
    display.displayMessage("Looking in the wordList");
    showListContents();

    display.displayMessage("\"updating\" the wordList and example array");
    for (String word : wordList)
    {
        word = "random silly phrase";
        word = null;
    }

    for (APSample sample : example)
    {
        sample = null;
    }

    display.displayMessage("Looking in the wordList");
    showListContents();
}
```

Say YES

```
private void showForEachCanChangeWithAccessors()
{
    showSamples();

    for (APSample current : example)
    {
        int random = (int) (Math.random() * 50);
        current.setValue("" + random);
    }

    showSamples();

}
```