

# Variables Revisit

## Chapter 6 - Lecture Slides

Naming, data types, instance variables, math and conversions

# Variables

## Variables

- **Variable** is data that is referenced by a named identifier
- Variables need to be **declared** before you use it
- Variable declaration includes:
  - 
  -
- Examples:
  - 
  -

## Variables - names

- Descriptive of the data it represents
- Using the alphabet is not acceptable: `a, b, c, d, e`; nor is `a1, a2, a3, a4`
- Lists of images in something like a slideshow can be named `img1, img2, img3`, etc.
- No spaces allowed within a variable name
- Convention is to begin variable names with a lower case letter and separate each word in the variable name by capitalizing subsequent words:
  - `firstName`,
- Abbreviations are good, but be consistent:
  - `tempFreq, hitFreq, qtyCount, qtyOfShirts`
- Cannot use a Java keyword for a variable name

# Java Keywords

- Reserved words
- Not to memorize, but take notice; which ones do you already recognize?

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

# Data Types

- There are 8 **primitive data types**
- We will only be concerned with the most common ones:
- Note that these are the primitive data types; we also use a lot of objects as well.
- Examples of objects are: JButton,

	data type	description	number of bits used to represent the number
integers	byte	Byte-length integer	8-bit
	short	Short integer	16-bit
	int	Integer	32-bit
	long	Long integer	64-bit
reals	float	Single-precision floating point	32-bit
	double	Double-precision floating point	64-bit
	char	A single character	16-bit Unicode character
	boolean	holds either the value <code>true</code> or <code>false</code>	1-bit

# Characters

- Data type:
- Enclose with single quotes
- Escape sequences

# Boolean

- One of two values
  - 
  -

# Variables

## ■ Instance variables

- ❑ Declared at
- ❑ Data available

## ■ Local variables

- ❑ Declared
- ❑ Data only available within
- ❑ Includes method parameters

# Instance vs. Local Variables

```
import java.awt.*;
import javax.swing.*;
public class instanceVars extends JApplet
{
    // instance variables
    JLabel name;
    JButton go;

    public void init()
    {
        setLayout( new FlowLayout( ) );
        name = new JLabel( "Cookie Monster" );
        go = new JButton( "GO!" );
        add( name );
        add( go );
    }
}
```

```
import java.awt.*;
import javax.swing.*;
public class instanceVars extends JApplet
{
    public void init()
    {
        // local variables
        JLabel name;
        JButton go;

        setLayout( new FlowLayout( ) );
        name = new JLabel( "Cookie Monster" );
        go = new JButton( "GO!" );
        add( name );
        add( go );
    }
}
```

# Scope

## ■ Scope

- ❑ Check out the enclosing squiggly
- ❑ You cannot declare two variables of the same name at the same scope level
- ❑ Any variable declared strictly within the scope of another with the same name will shadow the outer named variable

# Instance Variables

- ⊗ Most variables are declared at the top of the class – called instance variables

```
public class Fun extends JApplet
{
    // instance variables
    JButton b_submit ;
    JTextField tf_state;

    // methods
    public void init()
    {
        b_submit = new JButton( "Submit" );
        ...
    }
}
```

# Instance Variables

- ⌘ We do this so we can reference them throughout the program – called **scope**

```
public class Fun extends JApplet implements ActionListener
{
    // instance variables
    JButton b_submit;
    JTextField tf_state;
    // methods
    public void init()
    {
        b_submit = new JButton( "Submit" );
        tf_state = new JTextField( "great", 10 );
        doLeftSide( );
    }
    public void doLeftSide( )
    {
        JPanel p = new JPanel( new FlowLayout( ) );
        p.add( b_submit );
        p.add( tf_state );
    }
}
```

(c) 2005-2008 by Elizabeth Sugar Boese

13

# Instance Variables

- ⌘ A problematic example follows: **What's wrong and what happens?**

```
public class Fun extends JApplet implements ActionListener
{
    // methods
    public void init()
    {
        JButton b_submit = new JButton( "Submit" );
        JTextField tf_state = new JTextField( "great", 10 );
        doLeftSide( );
    }
    public void doLeftSide( )
    {
        JPanel p = new JPanel( new FlowLayout( ) );
        p.add( b_submit );
        p.add( tf_state );
        add( p );
    }
}
```

(c) 2005-2008 by Elizabeth Sugar Boese

14

# Math

# Math

Operator	Java code	Description
+	op1 + op2	Adds op1 and op2; also used to concatenate strings
-	op1 - op2	Subtracts op2 from op1
*	op1 * op2	Multiplies op1 by op2
/	op1 / op2	Divides op1 by op2
%	op1 % op2	Computes the remainder of dividing op1 by op2

- Math operators work as you would expect, with two that may be new to you:
- In Java, addition also works on strings by concatenating them together.
  - `String fullname = "Elizabeth" + "Boese";`  
fullname is equal to
  - `String result = "Version" + 2.0;`  
result is equal to

(c) 2005-2008 by Elizabeth Sugar Boese

16

## Math

Operator	Java code	Description
+	op1 + op2	Adds op1 and op2; also used to concatenate strings
-	op1 - op2	Subtracts op2 from op1
*	op1 * op2	Multiplies op1 by op2
/	op1 / op2	Divides op1 by op2
%	op1 % op2	Computes the remainder of dividing op1 by op2

- The modulus operator % returns the int remainder after dividing op1 by op2
  - int result = 9 % 3;
  - result is equal to \_\_\_\_\_
- When would the use of the modulus function be useful? When do you use this technique in your daily life?

## Java arithmetic

- Integer division
  - Throw away the remainder!
  - $9 / 4 =$
- Modulus
  - $9 \% 4 =$
  - How would you test to see if the int variable named **val** is even?
- Widening conversions
  - $9.0 / 4 =$

## Math class

- Can use the constants and methods from the Math class
  - Math.PI
  - Math.pow( double x, double y )
  - Math.round( double d )
  - Math.sqrt( double d )

## Mathematical Expressions

- Express the following in Java:

$$\frac{1}{\text{time} + 3\text{mass}}$$

$$\text{rate}^{50} + \text{amount}$$

# Relational Operators

Operator	Use	Description
>	op1 > op2	Returns true if op1 is greater than op2
>=	op1 >= op2	Returns true if op1 is greater than or equal to op2
<	op1 < op2	Returns true if op1 is less than op2
<=	op1 <= op2	Returns true if op1 is less than or equal to op2
==	op1 == op2	Returns true if op1 and op2 are equal
!=	op1 != op2	Returns true if op1 and op2 are not equal

- Relational operators return either **true** or **false** based on the operands
- Note that the greater-than-or-equal-to operator has the = AFTER the > and the less-than-or-equal-to operator has the = AFTER the < This is Java, so on an exam do not try to put ≤ or ≥ as our keyboards don't have these symbols!!! You have been Warned.
- Note the difference between the two equal signs here and the assignment statement that contains only one equal sign.
- Examples:

# Logical Operators

Trials: when x = -3, when x = 5, when x = 6

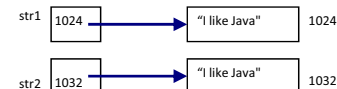
<	Less than	x < 5	
<=	Less than or equal to	x <= 5	
==	Equal	x == 5	
!=	Not equal	x != 5	
>=	Greater than or equal to	x >= 5	
>	Greater than	x > 5	

# Careful testing Equality!!

- Two primitive values are the same under == if their values are the same
- For two class-type variables, == is true ONLY if the variables REFER TO THE SAME OBJECT!

# Careful testing Equality!!

- Two primitive values are the same under == if their values are the same
- For two class-type variables, == is true ONLY if the variables REFER TO THE SAME OBJECT!



## Conditional Operators

Operator	Use	Description
&&	op1 && op2	Returns true if op1 and op2 are both true
	op1    op2	Returns true if either op1 or op2 is true
!	!op	Returns true if op is false

- Conditional operators return either **true** or **false** based on **boolean** operands
- Examples

## Truth Tables

a	b	!a	!b	a&&b	a    b	!a&&b	a    (b && !a)
F	F						
F	T						
T	F						
T	T						

## The wonderful instanceof Operator

- Test whether a variable is a particular *class* type by using the instanceof operator
- Variable instanceof class-type
  - B instanceof JButton
  - B instanceof MyOwnButton
  - Mb instanceof JFrame
- Can not check if something is an instance of a primitive data type!

## Expressions

- Precedence “order of operations”
  - Parenthesis first
  - unary operators (pos/neg, ++ -- if before variable, !)
  - \*, /, %
  - +, -
  - < <= > >= instanceof
  - == !=
  - &&
  - ||
  - =
  - $3 * ( 2 + ( 3 - 4 * 2 + (5-1) ) ) =$

# Conversions

# String to numbers

- Call methods on the class (Integer, Double, Character, etc.)

<u>Class:</u>	<u>Method</u>
Integer	parseInt( String )
Double	parseDouble( String )

- Examples**  
int value =  
double value =  
int numShirts =

# Numbers to String

- Call method from the String class:
- Examples  
To set the text inside a JTextField, we have to send it a String (not an int or double). So we can convert a number to a String with String.valueOf( number )

```
tfcartTotal.setText(           );
```

- What's another 'hack' way to convert a number to a String?  
append the number to an empty String (2 dbl-quotes) → String

```
tfcartTotal.setText(           );
```

# Summary

- Variables
  - Names
  - Java keywords
  - Data types
  - Scope
  - Instance vs. local variables
- Math
  - Arithmetic
  - Relational operators
  - Logical
  - Equality
  - Conditional
  - Order of precedence
- Conversions
  - String to number
  - Number to String