

Program Structure

comment	// header
libraries	imports; // explain why if possible
class definition	public class Lab1MainClass
	{
shared variables →	public variables;
main method →	public static void main(String args [])
	{
not-shared →	int x;
	// end main method
	}
support method	public static supportMethod()
	{
	// end supportMethod
	}
note end brackets	} // end Lab1MainClass

// header

- Name, Lab, Assumptions
- Comments :

/*

Block Comments

*/

// end-line comments

imports

```
import javax.swing.JOptionPane;
```

better than

```
import javax.swing.*;
```

no restrictions on import libraries in lab

public variables

- variables that are directly accessible from outside the class
- regarded as “unsafe”
- always initialize;

```
public static String userInput = “ ”;
```

private variables

- only accessible by methods within the class
- considered “safe”, because methods must be written in the class, to present these values to objects outside the class (Get/Set methods)
- always initialize

```
private int tempGrade = 0;
```

local variables

- Known only within the method in which they are defined. Sometimes called “trash” variables.

```
public static void main(String args [ ])  
{  
  int x;  
} // end main method
```

Variable “domain”

```
public class myMainClass  
{  
  public static int x;  
  private int y;  
  public static void main(String args [ ])  
  {  
    int z;  
    ..... code here .....  
  } // end main method  
  
  public void supportMethod( )  
  {  
    .....can x, y, or z be used here? .....  
  }  
} // end myMainClass
```

```
public class Lab1MainClass  
{  
  /*  
  contains main method  
  
  if possible: all operator input and operator  
  input support in this class  
  
  fundamental program behavior  
  */  
} // end Lab1MainClass
```

```
public static void main (String args [ ])
{
  int choice
  do {
    choice = JOptionPane.showConfirmDialog (null, "Exit?");
    if ( choice == 0 ) // only exit if == 0, or yes
      {
        System.exit(0);
      }

    // program work done here
  } while( true );
} // end main method
```

What a Class does

- A collection of methods and variables.
- Every Object created from a class, operates as an independent, freely running program.
- Only one main allowed
- main "spawns" useable objects, sometimes many objects of the same class.

An example

- A Class that controls video game character behavior (droidClass)
- droidClass instantiates armClass twice into two armObjects, legClass twice into two legObjects, headClass into headObject
- Each Class/Object defines unique behavior.
- Game (i.e. main) instantiates many droidClass objects, each one a game character.

"The Object Model"

- Programs are easy: they put the right information in the right place at the right time.
- Program design involves solving real world problems: control a robot, run a payroll department, display air traffic.
- There is a direct correlation between the parts of the program (objects) and the real world systems they are meant to model.

What we know about Objects

"Classes are of no value until..."
They are instantiated

Classes

- Collections of methods and variables
- You write the classes to do real work

Objects

- They are “instantiated” from Classes
- An “object”, is a Class that has been put to work.
- Since Classes can be used many times in the same program, each “instance” of a Class is a separate, usable copy.

Copies of Classes

- “Copy” is an oversimplification, but the concept has “run-time” value
- A copy is created using the `new` statement.
- A “copy” of a Class is an Object.
- non-Static Classes can never be used directly.
- They must be copied to Objects
- The copy step is called *instantiation*.
- Your program “owns” and controls the copy.

Instantiation

- Once a class is defined, “making it your own” for use, using the `new` keyword:

```
myClass myObject = new myClass( );
```

Creates an “instance” of the Class.
the Class solves a “Class” of problems.

Applications - The main Method

- The computer runs it, you don't.
- The computer calls the main method
- The main method uses methods, contained in Objects, derived from Classes that you write.

```
public static void main(String [ ] args)
{
    ShowColors app = new ShowColors( );

    // any method in ShowColors is now useable
    // as app.MethodName( );
}
```

Method calls and returns,
across classes

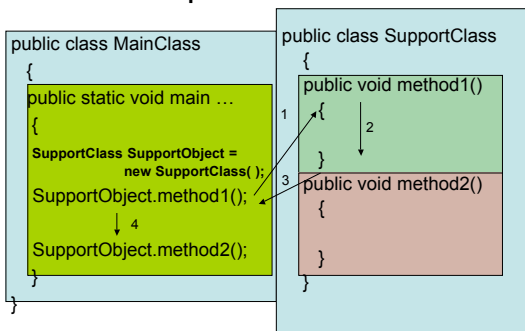
What is a method?

- Java code
- with a name
- and a declaration statement
- with brackets
- in a class

2 golden rules

1. Methods do not know about variables or values from outside the method, unless they are passed as parameters in the parenthesis.
2. Main does not know about the result of a method unless it is placed in the method's return statement, and the method is called from main using an assignment

The concept of "flow of control"



let's define counterClass

Define a class that will contain useful support methods, *to be used by other methods in other classes*.

Define one method in counterClass, called intervalSummer, that takes two integers, sums up all the numbers between the integers, and returns the result.

defining counterClass

```
public class counterClass
{
    // methods can only go in classes
}
```

Questions to ask yourself:

- will the method return a variable?
 - if no, then the define the method
public static void *methodname*
- if yes, what type of variable will it return?
 - if an int, define the method
public static int *methodname*
 - if a double, define the method
public static double *methodname*

Questions to ask yourself

- Will the method accept, and use incoming variables?
 - if yes, then give them names and place them in the parenthesis

```
public class counterClass
{
    public static int intervalSummer( int a, int b)
    {
        int total;
        return (total);
    } // end intervalSummer method
}
```

1st variable 2nd 3rd

Complete the method

```
public class counterClass
{
    public static int intervalSummer( int a, int b)
    {
        int total = 0;
        int x;
        for (x=a; x<=b; x++)
        {
            total = total + x;
        }
        return (total);
    }
}
```

values come into method

result of calculation leaves the method

- What will the method call look like?

```
result = counterObject.intervalSummer(4,10);
```

Using the method from main

```
public class myMainClass  
{  
  
  
  
  
  
  
}
```

```
public class myMainClass  
{  
public static void main( String [] args )  
{  
    int result;  
    counterClass counterObject = new counterClass( );  
  
    result = counterObject.intervalSummer( 4, 10 );  
    ↑                               ↓ ↓  
comes from return(total)         goes to a and b  
} // end main method  
}
```

across the main/class boundary

```
result = counterObject.intervalSummer( 4, 10 );
```

```
public static int intervalSummer( int a, int b)
{
    int total = 0;
    int x;
    for (x=a; x<=b; x++)
    {
        total = total + x;
    }
    return (total);
}
```

main (points to the call) | *counterClass* (points to the method definition)

```
① result = counterObject.intervalSummer( ② 4, ③ 10 );
```

```
public static int intervalSummer( int ④ a, int ⑤ b)
{
    int total = 0;
    int x;
    for (x=a; x<=b; x++)
    {
        total = total + 1;
    }
    return (total);
}
```

moving from literals to variables

```
public static void main( String [] args )
{
    int result;
    counterClass counterObject = new counterClass( );
    int x, y;
    x = 4;
    y = 10;
    result = counterObject.intervalSummer( x, y );
    System.out.println( result );
} // end main method
```

get x from user

```
public static void main( String [] args )
{
    int result;
    counterClass counterObject = new counterClass( );
    int x, y;
    String userInput;
    y = 10;
    userInput = JOptionPane.showInputDialog("Enter x:");
    x = Integer.parseInt( userInput );
    result = counterObject.intervalSummer( x, y );
    System.out.println( result );
}
```

get y from the user

```
public static void main( String [] args )
{
    int result;
    counterClass counterObject = new counterClass( );
    int x, y;
    String userInput;
    userInput = JOptionPane.showInputDialog("Enter x:");
    x = Integer.parseInt( userInput );
    userInput = JOptionPane.showInputDialog("Enter y:");
    y = Integer.parseInt( userInput );
    result = counterObject.intervalSummer( x, y );
    System.out.println( result );
}
```

method now has great utility

```
public static void main( String [] args )
{
    int result;
    counterClass counterObject = new counterClass( );
    int x, y;
    userInput = JOptionPane.showInputDialog("Enter x:");
    x = Integer.parseInt( userInput );
    userInput = JOptionPane.showInputDialog("Enter y:");
    y = Integer.parseInt( userInput );
    result = counterObject.intervalSummer( x, y );
    System.out.println( result );
}
```

Summary

1. Methods do not know about variables or values from outside the method, unless they are passed as parameters in the parenthesis.
2. Main does not know about the result of a method unless it is placed in the method's return statement, and the method is called from main using an assignment

When to write a method

- When you want to isolate something that...
 - is useful under many conditions
 - that you want to use over and over again
 - that will perform the same operation for you dependably
- example – a method that will repeatedly ask the user for an integer, until a proper integer is entered (Lab 7)

```
public static int getAnInteger ( )
{
    int x;
    String userInput;
    do { // do forever
        userInput = JOptionPane.showInputDialog("Enter an integer:");
    } try {
        x = Integer.parseInt( userInput );
        return ( x );
    } catch( Exception error ) {
        System.out.println( "Bad format, try again" );
    } while(true);
} // end method getAnInteger
```
