Instantiable Classes

It is not acceptable to write a program using just the main method. Why?

```
Programs become too large and unmanageable
```

It is not acceptable to write programs using just predefined classes. Why?

```
Predefined classes alone cannot satisfy all
programming needs
```

An important aspect of Object Oriented Programming is the ability to define instantiable classes. What are instantiable classes?

```
Classes from which instances (i.e., objects)
can be made
```

(NOTE: Instantiable classes are defined in order to define how an object of the class behaves)

Examples:

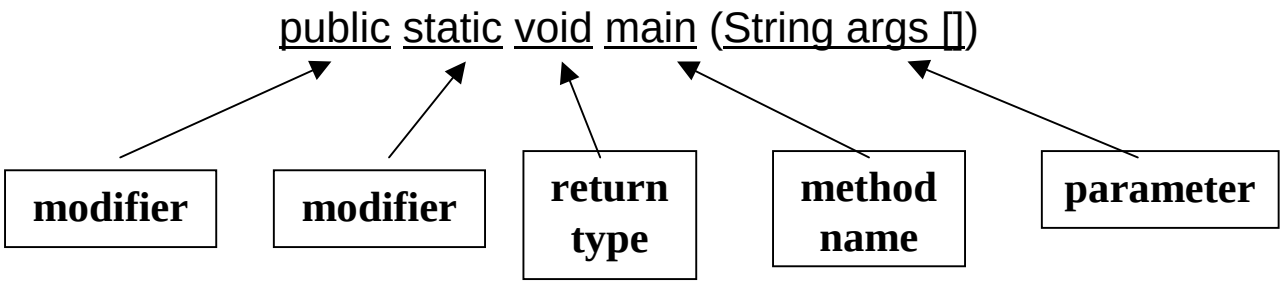
Instantiable Classes	Non-instantiable Classes
InputBox	Math
OutputBox	DrawingProgram
MainWindow	AreaCalc
Calculator	VolumeCalc
Mouse	FunTime

Creating instantiable classes is similar to creating non-instantiable classes. (See Template for a Class Definition)

Instantiable Classes...continued

❖ Class Members: the data values and methods defined within a class

- data values
  - also known as “data members”
  - can be instance or class data values
  - can be variables or constants
- methods
  - recall method declaration
  - method header comment includes:
    1. method name
    2. purpose
    3. parameters
    4. return values



Constructors

❖ A "special" class method that is called when the 'new' message is sent to a class (i.e., when a new object of that class is created)

❖ The purpose of the constructor is to initialize the data members of the newly created object (and to designate memory space for the object)

❖ Syntax:

```
public <class name> (<parameters>){
    <statements>
}
```

NOTES:

- The name of the constructor is the same as the name of the class
- There is no return type specified in the declaration
- There is no "static" modifier even though the constructor is considered to be a class method (static is implied)
- The visibility is almost always public

❖ More than one constructor can be defined for the same class

- Each constructor would have a different set of parameters
- Allows for the creation of instances of the class in different ways
  - Different number of parameters
  - Different data types for the parameters

- Example:

```
public MainWindow () {
    Title = "Main Window";
}

public MainWindow (String s) {
    Title = s;
}
```

Constructors...continued

- If NO constructor is defined for a class, the compiler will create a default constructor that does nothing except allocate memory for the newly created object.

- A good programmer ALWAYS defines a constructor

- A constructor should initialize ALL data members to some value (This insures that the object is initialized to a “valid state”)

Visibility Modifiers

public: for all to see and use
("outside" methods have access)

private: only seen/used by other class members
("outside" methods are denied access)

❖ Guidelines in determining the visibility of data members and methods

1. Declare the class and instance variables **`private`**
2. Declare the class and instance methods **`private`** if they are used only by the other methods in the same class. Otherwise, declare them **`public`**.
3. Declare the class constants **`public`** if you want to make their values directly readable by outside methods. If the class constants are used for internal purposes only, then declare them **`private`**.

❖ With data members private, their value can only be changed via the public methods defined in the class, thus insuring the integrity of the class.

❖ Data members are considered implementation details of the class, and they should be kept invisible from the outside by declaring them private.

❖ **`private`** hides the "internal" details and allows indirect access only (analogy: using remote/buttons on TV or VCR to change channel or program)

Static Modifier

- Instance methods are declared WITHOUT the static modifier.
- Class methods are declared WITH the static modifier.

- Instance variables are declared WITHOUT the static modifier.
- Class variables are declared WITH the static modifier.

- Class methods can access ONLY class variables and constants, but not instance variables.
- Instance methods can access ALL variables and constants.

- The main method is ALWAYS static.

- (NOTE: Constructors are by default static even though no static modifier is included in the declaration)

STATIC: One version of the method and/or data value for the WHOLE class and ALL objects made from the class

NON-STATIC: One version of the method and/or data value for EACH object made from the class

```
class MainClass {

    private int data ;

    /* CONSTRUCTOR */
    public MainClass () { data = 0; }

    public static void main (String args []) {
        MainClass mc = new MainClass ();

        staticMethod ();               // OK
        MainClass.staticMethod ();     // OK--Better
        mc.staticMethod ();            // OK

        nonStaticMethod ():            // NOT OK
        MainClass.nonStaticMethod (); // NOT OK
        mc.nonStaticMethod ();         // OK
    }

    public static void staticMethod () {
        staticMethod ();               // OK
        MainClass.staticMethod ();     // OK--Better
        mc.staticMethod ();            // NOT OK (No mc)

        nonStaticMethod ():            // NOT OK
        MainClass.nonStaticMethod (); // NOT OK
        mc.nonStaticMethod ();         // NOT OK (No mc)
    }

    public void nonStaticMethod () {
        staticMethod ();               // OK
        MainClass.staticMethod ();     // OK--Better
        mc.staticMethod ();            // NOT OK (No mc)
        this.staticMethod ();          // OK

        nonStaticMethod ():            // OK
        MainClass.nonStaticMethod (); // NOT OK
        mc.nonStaticMethod ();         // NOT OK (No mc)
        this.nonStaticMethod ();       // OK
    }
}
```

Local Variables

- Definition: a variable declared within the method declaration
- Local variables are accessible only from the method in which they are declared and they are available only when the method is being executed (local variables are erased when the execution of a method is completed)
- Parameters are also considered local variables

Return Values

- Syntax for return statement:

```
return <expression>;
```

- The result value of the return expression must be assignment compatible with the return type specified in the method declaration

```
private float intRate;

public float getInterestRate () {
    return intRate;
}
```

- Because a value is returned from the method, the method call (i.e., message) can be used anywhere an expression can appear.

```
public double getSqrt (double d) {
    return Math.sqrt (d);
}
```

Message Sending

Sender

- Calls a method by sending a message
- Message must meet the expectations of the receiver

Receiver

- Executes a method by receiving a message
- The message must match the method definition

Assume you have the following variables:

```
int x = 9;
double d = 22.5;
MainWindow mw = new MainWindow ();
Object o = new Object ();
```

	Messages
1	`o.setValues (11.0, 9, x);`
2	`o.setValue (11.0, x);`
3	`x = setValues (d);`
4	`o.setValues (mw, x);`
5	`d = o.setValues (11.0, 9, x);`
6	`o.setValues (d, 9, 11);`
7	`o.setValues (11, 9, x);`
8	`x = o.setValues (11.0, d, x);`
9	`o.setValues (d, x);`

	Object class method definitions
1	`public void setValues (double d, int x, int y)`
2	`public void setValues (double d, int y)`
3	`public int setValues (double y)`
4	`public void setValues (MainWindow m, int x)`
5	`public int setValues (double d, int x, int y)`
6	`public void setValues (double d, double, x)`
7	`public int setValues (int x, int y, int z)`
8	`private void setValues (double d, int x, int y)`
9	`public void setValues (int d, int x)`

1-1 1-5 6-7 3-3 7-7 7-1 2-9

Arguments & Parameters

1. Arguments are passed to a method using the pass-by-value scheme

2. Arguments are matched to the parameters from left to right. The data type of an argument must be assignment compatible to the data type of the matching parameter

3. The number of arguments in the method call must match the number of parameters in the method definition

4. Parameters and arguments do not have to have the same name

5. Local copies, which are distinct from arguments, are created even if the parameters and arguments share the same name

6. Parameters are input to a method, and they are local to the method. Changes made to the parameters will not affect the value of corresponding arguments (EXCEPTION: objects passed as arguments)

"arguments" are also known as "actual parameters"

"parameters" are also known as "formal parameters"