Homework HW1

The answers you handed in at the end of lecture 1 showed mass confusion! Perhaps 80% of you weren't sure what to write. This was not graded! It was only to help us and you assess the situation.

Doing HW1 will eliminate the confusion. Piazza note @22, (find a link to it in the pinned Piazza Recitation/Homework note.)

Evaluation, Execution, Syntax, Semantics.
Presenting an algorithm in English (2.5 minutes).
Executing the assignment statement (2.5 minutes).
Do HW1 and submit on the CMS

Homework

1. Study material of this lecture.
2. Visit JavaHyperText, click on Code Style. Study
   3. Documentation
      3.1 Kinds of comments
      3.2 Don't over-comment
      3.4 Method specifications
      3.4.1 Precondition and postcondition
3. Spend a few minutes perusing slides for lecture 3; bring them to lecture 3.

Java OO

Python and Matlab have objects and classes.
Strong-typing nature of Java changes how OO is done and how useful it is. Put aside your previous experience with OO (if any).

This lecture:
First: describe objects, demoing their creation and use.
Second: Show you a class definition, a blueprint for objects, and how it contains definitions of methods (functions and procedures) that appear in each object of the class.
Third: Talk about keyword null.
Fourth: Introduce Exceptions

References to JavaHyperText entries

Objects: object
Calling methods: method call
Class definition: class def
public, private: public private method
Parameter vs argument: parameter, argument
Inside-out rule
Methods may have parameters
Method calls may have arguments

Fields of an object may be mentioned. We cover these in next lecture.

Function: a method that returns a result.
Procedure: method that does not return a result, void method.
**Drawing an object of class javax.swing.JFrame**

This object is associated with a window on your computer monitor.

Name of object, giving class name and its memory location (hexadecimal 25c7), Java creates name when it creates object.

Object contains methods (functions and procedures), which can be called to operate on the object.

Function: returns a value; call on it is an expression.

Procedure: does not return a value; call on it is a statement.

**Evaluation of new-expression creates an object**

Evaluation of `new javax.swing.JFrame()` creates an object and gives as its value the name of the object.

If evaluation creates this object, value of expression is `JFrame@25c7`.

**A class variable contains the name of an object**

Type `JFrame`: Names of objects of class JFrame.

If evaluation of `new-expression` creates the object shown, name of object is stored in `h`.

Consequence: a class variable contains not an object but name of an object, pointer to it. Objects are referenced indirectly.

**Class definition: a blueprint for objects of the class**

Class definition: Describes format of an object (instance) of the class.

```
/** A description of what the class is for */

public class C {
    // declarations of methods (in any order)
}
```

This is a comment.

Access modifier `public` means C can be used anywhere.

Class definition C goes in its own file named C.java.

On your hard drive, have separate directory for each Java project you write; put all class definitions for program in that directory. You’ll see this when we demo.

**First class definition**

```java
/** An instance (object of the class) has (almost) no methods */

public class C {
}
```

Then, execution of

```java
C k;
k = new C();
```

creates object shown to right and stores its name in `k`.

**A class variable contains the name of an object**

```
h = new javax.swing.JFrame();
```

If variable `h` contains the name of an object, you can call methods of the object using dot notation:

Procedure calls:

```java
h.show();
h.setTitle("this is a title");
```

Function calls:

```java
h.getX();
h.getX() + h.getWidth();
```

```java
x = y;
g = h;
```
Class extends (is a subclass of) JFrame

/** An instance is a subclass of JFrame */
public class C extends javax.swing.JFrame {
    C: subclass of JFrame
    JFrame: superclass of C
    C inherits all methods that are in a JFrame
    C@6667
    hide(), show(), setTitle(), getLocation(), getLocation(), getWidth(), getHeight() ...

    Easy re-use of program part!
}

Class definition with a function definition

/** An instance is a subclass of JFrame with a function area */
public class C extends javax.swing.JFrame {
    /** Function area */
    public int area() {
        return getWidth() * getHeight();
    }
    C@6667
    ... getWidth(), getHeight() ...

    Function calls automatically call functions that are in the object

    JFrame
    C@6667
    getWidth(), getHeight() ...
    area() {
        return getWidth() * getHeight();
    }

    JFrame
    C@2abc
    getWidth(), getHeight() ...
    area() {
        return getWidth() * getHeight();
    }

    JFrame
    …

    JFrame
    C@6667
    getWidth(), getHeight() ...
    area() {
        return getWidth() * getHeight();
    }

    JFrame
    C@2abc
    getWidth(), getHeight() ...
    area() {
        return getWidth() * getHeight();
    }

Inside-out rule for finding declaration

/** An instance ... */
public class C extends javax.swing.JFrame {
    /** Return area of window */
    public int area() {
        return getWidth() * getHeight();
    }
    C@6667
    getWidth(), getHeight() ...
    area() {
        return getWidth() * getHeight();
    }

    The whole method is in the object

    JFrame
    C@6667
    …
    area() {
        return getWidth() * getHeight();
    }

    JFrame
    C@2abc
    …
    area() {
        return getWidth() * getHeight();
    }

Inside-out rule for finding declaration

/** An instance ... */
public class C extends javax.swing.JFrame {
    /** Return area of window */
    public int area() {
        return getWidth() * getHeight();
    }
    C@6667
    getWidth(), getHeight() ...
    area() {
        return getWidth() * getHeight();
    }

    Function area in each object. getWidth() calls function getWidth in the object in which it appears.

    JFrame
    C@6667
    …
    getWidth(), getHeight() ...
    area() {
        return getWidth() * getHeight();
    }

    JFrame
    C@2abc
    …
    getWidth(), getHeight() ...
    area() {
        return getWidth() * getHeight();
    }

Class definition with a procedure definition

/** An instance is a JFrame with more methods */
public class C extends javax.swing.JFrame {
    public int area() {
        return getWidth() * getHeight();
    }
    C@6667
    …
    …
    …

    JFrame
    C@6667
    …
    …
    …

    JFrame
    C@2abc
    …
    …
    …

Inside-out rule for finding declaration

/** An instance ... */
public class C extends javax.swing.JFrame {
    /** Set width of window to its height */
    public void setSize() {
        setSize(getHeight(), getHeight());
    }
    C@6667
    …
    setSize(int, int), getHeight(), getHeight() ...
    setSize(getHeight(), getHeight());

    It is a procedure because it has void instead of return type

    JFrame
    C@6667
    …
    setSize(int, int)
    getHeight(), getHeight() ...
    setSize(getHeight(), getHeight());

    JFrame
    C@2abc
    …
    setSize(int, int)
    getHeight(), getHeight() ...
    setSize(getHeight(), getHeight());

Using an object of class Date

/** An instance is a JFrame with more methods */
public class C extends javax.swing.JFrame {
    …
    /** Put the date and time in the title */
    public void setTitleToDate() {
        setTitle(new java.util.Date().toString());
    }
    C@6667
    …
    setTitle(new java.util.Date().toString());

    An object of class java.util.Date contains the date and time at which it was created.
    It has a function toString(), which yields the data as a String.

    JFrame
    C@6667
    …
    setTitle(new java.util.Date().toString());

    JFrame
    C@2abc
    …
    setTitle(new java.util.Date().toString());
**About null**

- v1
  - C@16
  - getName()
- v2
  - null

null denotes the absence of a name.

v2.getName() is a mistake! Program stops with a NullPointerException.

You can write assignments like: v1 = null;
and expressions like: v1 == null

**Intro to Exceptions**

```java
int x = 5;
System.out.println("x is now "+x);
assert x == 6;
```

When the assert statement is executed and x is not 6, an object of class AssertionError is created and “thrown”. It contains info needed to print out a nice message.

```java
java.lang.AssertionError
at A0.main(A0.java:9)
```

**Intro to Exceptions**

```java
public static void m() {
    int y = 5/0;
}
```

When 5/0 is evaluated, an object of class ArithmeticException is created and “thrown”. It contains info needed to print out a nice message.

```java
java.lang.ArithmeticException
at A0.m(A0.java:15)
```

You will learn all about exceptions in next week’s recitation!