What we'll cover today

- What happens when we write a Java program?
- Definitions from the example in Monday's class
- What does a Java program look like?
- Core Java concepts

What happens when we write a Java program?

- Step 1: Write the program in a text editor
  - saved as human-readable text
- Step 2: compile
  - converts text into numerical code called bytecode
  - bytecode is a language that's understood by the Java interpreter
- Step 3: run the program
  - program is interpreted

What do we mean by “interpreted”?

- In most programming languages, programs are compiled into machine language
  - i.e., code that a particular OS (WindowsXP, Linux, Mac OSX) understands
  - machine language can only be understood by OSes of the same type (i.e., all WindowsXP machines, or all Mac OSX machines)
  - if we want to run a program on different OSes, we need to recompile the program on those OSes
Java is a bit different....

- Java is a “write-once, run-anywhere” language
  - it’s meant to run on any OS without recompiling it for each OS
  - uses a virtual machine that “translates” the code in terms that the OS can understand.
  - Java code is compiled once, then each OS interprets it
- This is a bit different from pure interpreted languages, like Perl, which are never compiled

Definitions from Monday’s class

- **Class**: definition or template of an object
  - class Cat
- **Object**: specific instance of a class
  - Cat fluffy;
- **Method**: action that a class performs
  - fluffy.purr();
- **Variable**: data used by a class (also called data member)
  - fluffy.numLegs;

What does a Java program look like?

- Example: SimpleShapes.java

Import statements

```java
import javax.swing.*;
import java.awt.*;

- Allow us to use predefined classes in Java by their “short names”
  - e.g., Graphics rather than java.awt.Graphics
- Appear at the top of the program
```
Comments

- Explain your logic to others
- "Document" your program
- MANDATORY in all assignments in this class!

Types of comments

- "Javadoc" style
  - Before class definitions
  - Before method definitions
  - Before class-wide variable declarations
    - /** ... */
    - Multi-line
- "C" style
  - /* ... */
  - Multi-line
- "C++" style
  - // ...
  - Single line

Use either style within methods to describe your logic

Class definition

class SimpleShapes extends JComponent {...}

- Gives the name of the class
- Describes its relationship to other classes (what it inherits from, etc., if applicable) --- extends keyword
- The name of the file must be the same as the class name!
  - i.e., this class will appear in the file SimpleShapes.java

Instantiating an object of a class

JFrame win;
win = new JFrame("Simple Shapes Example");

- Variables must be declared before they are used
  objectType objectName;
  dataType variableName;
- Objects must be created before they are used
  objectName = new ObjectType(parameters);
Variable declarations

- Variable declarations that appear at the top of a program (outside the methods) are variables that will be used by one or methods in the class
  - sometimes called global variables, because they can be seen “globally” (by all methods in the class)
- There are no global variables in SimpleShapes
- Variables can also be declared within methods
  - they will only be “visible” inside these methods

What happens when we run the preceding program?

- The first statement in doSomethingElse() is ok
- The second statement is not, because b only exists in doSomething()
  - b is “local” to doSomething()
  - a is “global” to class Foo
- The program will not compile and we will get an error

Example: variable visibility

```java
public class Foo {
    private int a;
    ...
    public void doSomething() {
        int b = 18;
        a = 36;
    }
    public void doSomethingElse() {
        System.out.println("a = "+a);
        System.out.println("b = "+b);
    }
}
```

Methods

```java
public void paint(Graphics g) {...}
```

- Methods are the actions that the class can perform
- Properties
  - arguments: data that is passed to the function from somewhere else externally
    - e.g., ...(Graphics g)
  - aka parameters
  - consist of a data type (Graphics) and a name (g)
  - some methods have no arguments
Methods: properties (continued)

- **return value**: the data type that the method returns to the place that called it
  - e.g., `public int ReturnAnInteger(...) returns an integer value`
  - some methods do not return anything, in which case the return value is `void`
- **visibility modifier**: determines who can access this method from outside the class
  - more on this much later
  - for now, assume most methods will be `public`

Calling methods in a class

```
win.setSize(500,500);
g.fillRectangle(50,50,300,150)
```

- Follows the format
  - `objectName.methodName(params)`
- Called “dot notation”

Special methods

- **Constructor**: “sets up” a class
  - initializes variables, etc.
  - this example does not have one
  - much more on this later!

- **Main**: makes the Java program executable
  - allows us to run the program
  - does **not** have access to class variables or methods directly!
    - must “instantiate” an object of the class first

More about `main()`

- `main()` is declared in a special way:
  - `public static void main (String[] args)`
  - `static` means that we can call this method outside of the class (i.e., we don't have to instantiate SimpleShapes to call main())
  - `void` means that it does not return anything
  - `public` means everyone can access it
  - `String[] args` means that the command line arguments to this program are stored in an array named `args`
What is a command line argument?

- A piece of data that we pass to a Java program on the command line
- Example:
  
  ```java
  java MyClass 3
  ```
  
  passes the value 3 to the `main` method of `MyClass`, and stores it in the array `args` at position 0
  
  ```java
  args[0] = 3;
  ```
  
  This value can now be used by `main()`

Conventions in this class

- `foo` refers to a variable
- `foo()` refers to a method
- `Foo` refers to a class
- `foo.bar()` refers to a method named `bar` in instance `foo` of class `Foo`

Good programming practices

- see web page

REMINDER

CLASS ON FRIDAY IS 1:50-3:00
IN CMC 306