Introduction to Java

CS1316: Representing Structure and Behavior

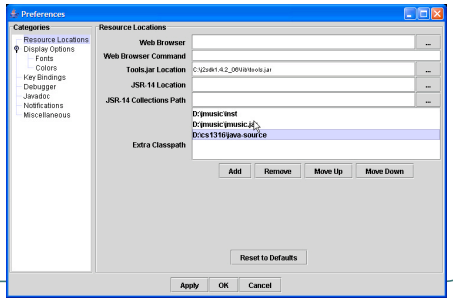
Story

- Getting started with Java
  - Installing it
  - Using DrJava
- Basics of Java
  - It's about objects and classes, modeling and simulation, what things *know* and what they can *do*.
  - Examples from Pictures, Sounds, and music.

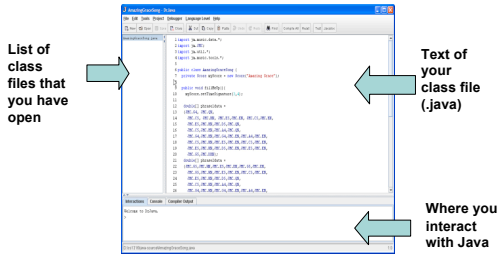
Things to do to get started

- Download and install JDK (Java Development Kit)
- Download and install DrJava
- Download JMusic
- Download the Java source files for class
- Then, tell Java where to find the JMusic and Java source files.

Telling DrJava where to find files



Parts of DrJava



Java is object-oriented

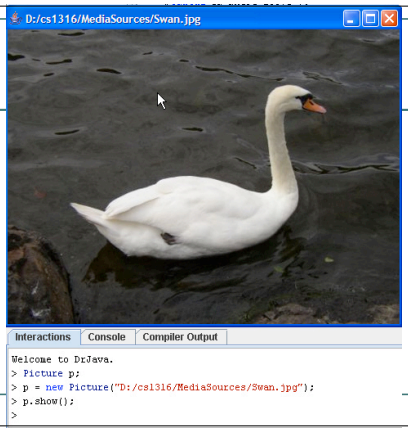
- 95% of everything in Java is an *object*.
- In *object-oriented programming*, you care about more than ***just*** specifying the process.
  - You care about who (or what) ***does*** the process,
  - And how the overall process *emerges* from the interaction of different objects.

Object-oriented programming is about modeling and simulation

- The whole idea of object-oriented programming is to create a *model* of the part of the world (real or imaginary).
- Creates constraints:
  - The real world doesn't have one set of rules/steps. (
    - You don't write one big program.
  - In the real world, no one *knows* everything, no one can *do* everything.
    - Each object has it's own things it knows and things it can do.

Variables in Java know their *types*

- Variables in Java know the kinds of things (values) they can hold.
- Objects in Java are organized into *classes*.
  - A class specifies what *all* the objects of that class *know* and can *do*.
  - All pictures can *show* themselves, even though each picture is different.
- Variables in Java are specific to particular *classes*.
  - We *declare* a variable to *only* hold objects of particular classes.



Did you get an error?

- If you got an error as soon as you typed **Picture p;** there are several possibilities.
  - All the Java files we provide you are in *source* form.
    - You need to compile them to use them.
    - Open **Picture.java** and click *Compile All*
  - You might not have your Preferences set up correctly. If Java can't find Picture, you can't use it.

Explaining what's going on

```
> Picture p;
> p = new
  Picture("D:/cs1316/
  MediaSources/Swa
  n.jpg");
> p.show();
```

- Every line ends with a semi-colon in Java.
  - (DrJava doesn't *always* require it.)
- *Picture* is the name of a class in Java.
- *p* is the variable that we're *declaring*

In Java programs, You can only use declared variables!

And you can only declare them once!

Explaining what's going on

```
> Picture p;
> p = new
  Picture("D:/cs1316/
  MediaSources/Swa
  n.jpg");
> p.show();
```

- *new Picture()* creates a new picture.
  - The *pathname* provided as an argument tells it *which* picture.
  - You can *always* use "/" and it'll *always* work (on any platform)
- *p* now *refers* to the new object (instance of class Picture)

Explaining what's going on

```
> Picture p;
> p = new
    Picture("D:/cs1316/
    MediaSources/Swa
    n.jpg");
> p.show();
```

- *Instances* of the class *Picture* (objects created from the class Picture) know how to *show* themselves.
- We access what the object *knows* and *can do* with the *dot operator.*
- *p.show()* says "Object that *p* refers to, would you please execute your *show() method*?"

Semicolons or not in DrJava Interactions Pane

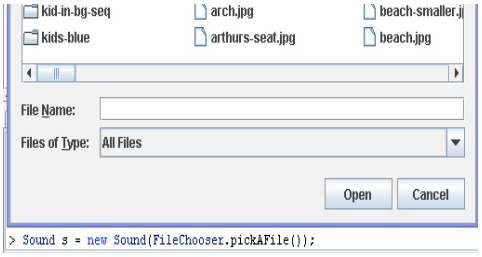
- No semi-colon says "Evaluate this, and show me the result."
- Semi-colon says "Treat this like a line of code, just as if it were in the Code Pane."

```
> p
Picture, filename
   D:/cs1316/MediaSources/Swan.jpg height 360
   width 480
> p;
>
```

Summarizing the terms so-far

- Just about everything in Java is an *object*
- Objects *know* specific things and can *do* specific things
  - Things they *know* are stored in variables (data)
  - Things they can *do* are grouped into *methods*
    - Think of *methods* as "functions known only to instances of that class."
- Objects are *instances* of a given *class* in Java.
  - All the instances know the same things and can do the same things.
- Variables are specific to a given class, and can only *refer* to objects of that type.

Let's create a sound



Explaining what's going on

```
> Sound s = new
    Sound(FileChooser.pickAFile());
> s.play();
```

- We can create an object *as* we declare the variable.
- *FileChooser* is an object that knows how to *pickAFile()* which puts up a file picker and returns a string.
- Instances of the class *Sound* know how to *play*, thus *s.play()*

What if we get it wrong?

```
> s.play();
> s.show();
Error: No 'show' method in 'Sound' with arguments: ()
> Picture.play();
Error: No 'play' method in 'Picture' with arguments: ()
> anotherpicture.play();
Error: Undefined class 'anotherpicture'
```

An example with Music

```
> import jm.util.*;
> import jm.music.data.*;
> Note n1;
> n1 = new Note(60,0.5);
> // Create an eighth note
  at C octave 4
```

• JMusic pieces need to be *import*ed first to use them.

An example with Music

```
> import jm.util.*;
> import jm.music.data.*;
> Note n1;
> n1 = new Note(60,0.5);
> // Create an eighth note
  at C octave 4
```

• Declare a *Note* variable.

An example with Music

```
> import jm.util.*;
> import jm.music.data.*;
> Note n1;
> n1 = new Note(60,0.5);
> // Create an eighth note
  at C octave 4
```

• *Note* instances have nothing to do with filenames.

• To create a note, you need to know *which* note, and a *duration*

Starting a line with // creates a *comment*—ignored by Java

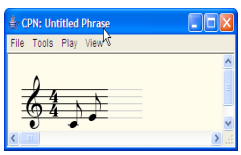
MIDI notes

Octave #	Note Numbers										
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

TABLE 2.1: MIDI notes

Making more notes

```
> Note n2=new Note(64,0.5);
> View.notate(n1);
Error: No 'notate' method in
     'jm.util.View' with arguments:
     (jm.music.data.Note)
> Phrase phr = new Phrase();
> phr.addNote(n1);
> phr.addNote(n2);
> View.notate(phr);
-- Constructing MIDI file
     from'Untitled Score'...
     Playing with JavaSound ...
     Completed MIDI playback ---
     -----
```



What's going on here?

```
> Note n2=new Note(64,0.5);
> View.notate(n1);
Error: No 'notate' method in
'jm.util.View' with arguments:
(jm.music.data.Note)
```

• We'll make another *Note* (at E4, another eighth note)

• There is an object named *View* that knows how to *notate* parts of music, but not an individual note.

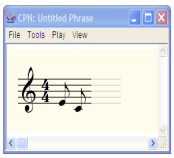
What's going on here?

```
> Phrase phr = new Phrase();
> phr.addNote(n1);
> phr.addNote(n2);
> View.notate(phr);
-- Constructing MIDI file from'Untitled Score'... Playing with JavaSound ... Completed MIDI playback --------
```

- We'll create a new *Phrase* instance and make a variable *phr* to refer to it. (*phr* has to be declared to be a *Phrase*.)
- *Phrase* instances know how to *addNote* notes to them. These are methods that take an argument—a *Note* instance.
- The *View* object does know how to *notate* an input *Phrase* instance. It generates this cool window where you see the notes and can play them (or save them as MIDI.)

Playing a different Phrase

```
> import jm.JMC;
> Phrase nuphr = new
    Phrase(0.0,JMC.FLUTE);
> nuphr.addNote(n2);
> nuphr.addNote(n1);
> View.notate(nuphr);
```



- We can specify when a phrase starts and with what instrument.
- We can add notes (even the same notes!) in different orders

Modeling Music

- The JMusic package is really *modeling* music.
  - Notes have tones and durations.
  - Musical Phrases are collections of notes in a sequence.
  - We can play (and View) a musical phrase.
    - A phrase doesn't have to start when other phrases do, and a phrase can have its own instrument.

Objects know things and can do things

	What instances of this class *know*	**What instances of this class *can do***
Note	A musical note and a duration	<Nothing we've seen yet>
Phrase	The notes in the phrase	addNote(aNote)