Today -- new Stuff!

• Two kinds of methods:
  – object
  – class

• Creating Classes
  – identifying objects and classes
  – constructors
  – adding a method, accessors and modifiers, creating a main method
  – comments, javadocs
A note about colors

• Exercises are colored “cantaloupe”.
  – please try some of these before Lab on Friday.
  – be sure to ask questions on Friday for anything you don’t understand

• Breaks between sections are blue.
Two kinds of methods

• Object methods
• Class methods
Object Methods

- So far we have created object methods
- Object methods must be invoked on an object
  - And they work on the object
    - Which is implicitly passed to an object method
      - And can be referenced using the ‘this’ keyword

- Examples
  - pictureObj.show();
  - soundObj.play();
  - turtleObj.forward();
Class (Static) Methods

• Can be invoked using the class name
  – Or invoked on an object
• Are used for general methods
  – Like Math.abs(num);
• Also used to create objects of the class
  – Sound s = Sound.createSineWave();
• Can only work with class (static) fields
  – Not object fields
• Are declared with the keyword static
  – Usually after the visibility
Create Sine Wave Class Method

```java
class SineWaveClass
{
    public static Sound createSineWave(int freq, int maxAmplitude)
    {
        String file = FileChooser.getMediaPath("sec1silence.wav");
        Sound s = new Sound(file);
        double samplingRate = s.getSamplingRate();
        double rawValue = 0;
        int value = 0;
        double interval = 1.0 / freq; // length of cycle in seconds
        double samplesPerCycle = interval * samplingRate;
        double maxValue = 2 * Math.PI;
        // Further code...
    }
}
```
// loop through the length of the sound
for (int i = 0; i < s.getLength(); i++)
{
    // calculate the value between -1 and 1
    rawValue = Math.sin((i / samplesPerCycle) * maxValue);

    // multiply by the desired max amplitude
    value = (int) (maxAmplitude * rawValue);

    // set the value at this index
    s.setSampleValueAt(i, value);
}

return s;
Testing Create a Sine Wave

• To create a sound with 880 Hz and a maximum amplitude of 4000:
  Sound s = Sound.createSineWave(880, 4000);
s.explore();
Invoking a Class Method Exercise

• Use the class method
  – `createSineWave(freq, maxAmplitude)`

• To create 4 sounds and save each to a file
  – Try `freq = 440` and `maxAmplitude = 8000`
  – Try `freq = 440` and `maxAmplitude = 10000`
  – Try `freq = 880` and `maxAmplitude = 8000`
  – Try `freq = 880` and `maxAmplitude = 10000`

• Try invoking the method on a sound object
  – Does this work?
Class Methods

- All methods are compiled and the code for the methods is stored in the object that defines the class
  - An object of the class called class
- Object methods must be called on an object
  - And the object is implicitly passed to the method
- Class methods can be called using the class name or on an object of the class
  - Objects always have a reference to their class

Sound Class: Class
Methods for the class
Class fields

s1: Sound
Object fields

s2: Sound
Object fields

based on slides by Barb Ericson,
Georgia Institute of Technology
Creating Square Waves

- To create a square shaped wave we can use positive and negative values
  - and switch between them at the halfway point in the cycle
- Pass in the desired frequency and maximum amplitude
- Calculate the interval (1 / freq)
- Calculate the samples per cycle
  - Interval * sampling rate
- Calculate half of the samples per cycle
- Loop through the whole sound
  - Init a sampleCounter to 0
  - If the sampleCounter < the halfSamplesPerCycle
    - Use maxAmplitude
  - Else
    - Use -1 * maxAmplitude
  - If the sampleCounter is >= samplesPerCycle reset it to 0

based on slides by Barb Ericson, Georgia Institute of Technology
Create Square Wave Exercise

• Write a class (static) method
  – createSquareWave(int freq, int maxAmplitude);
• It should generate and return a sound
  – With a length of 1 second
  – That is comprised of square waves
• Use it to generate
  – Try freq = 440 and maxAmplitude =  8000
  – Try freq = 440 and maxAmplitude =  10000
  – Try freq = 880 and maxAmplitude =  8000
  – Try freq = 880 and maxAmplitude =  10000
• Compare the square waves and sine waves
MidiPlayer Class

- Create an object of this class
  - MidiPlayer player = new MidiPlayer();
- Use it to play a note (with a duration)
  - player.playNote(62, 250);
    - The 62 is the key on the piano (d in fourth octave)
    - The 250 is the length of time to play it
      - out of 1000 milliseconds so if a measure is played in 1 second
        this is a quarter note
- Specify rests with
  - player.rest(int duration) in milliseconds
public void playJingleBells4() {
    // measure 1
    playNote(52,250); // e eighth note
    playNote(60,250); // c eighth note
    playNote(58,250); // b flat eighth note
    playNote(56,250); // a flat eighth note
    playNote(56,250); // a flat eighth note

    // measure 2
    playNote(52,500); // e quarter note
    rest(250); // rest
    playNote(52,125); // e sixteenth note
    playNote(52,125); // e sixteenth note
    playNote(52,125); // e sixteenth note

    // measure 3
    playNote(52,500); // e eighth note
    playNote(60,250); // c eighth note
    playNote(58,250); // b flat eighth note
    playNote(56,250); // a flat eighth note

    // measure 4
    playNote(53,1000); // f half note
}

based on slides by Barb Ericson,
Georgia Institute of Technology
Setting the Instrument

- Use
  - setInstrument(int number)
  - To set the instrument to make the sounds with

Testing playJingleBells4

```java
MidiPlayer player = new MidiPlayer();
player.setInstrument(MidiPlayer.FLUTE);
player.playJingleBells4();
```
Constants

- The instrument numbers are represented with class constants
  - in the MidiPlayer class
- A constant is something that doesn’t change
  - Declared with the keyword final
  - If you try to change it after the constructor is called you will get a compile error
- Class constants are variables that have space in the object that defines the class
  - Declared with the keywords static and final
  - Can be used by Class.Constant
- Java naming convention is to use all uppercase letters for constants
  - With _ between words
Some Instrument Constants

- MidiPlayer.PIANO
- MidiPlayer.MUSIC_BOX
- MidiPlayer.GUITAR
- MidiPlayer.HARP
- MidiPlayer.TROMBONE
- MidiPlayer.TRUMPET
- MidiPlayer.ALTO_SAX
- MidiPlayer.TENOR_SAX
Play a Song Exercise

• Write a method to play at least 4 measures of a song
• For public domain sheet music of classical piano see
  – http://www.sheetmusic1.com/NEW.GREAT.MUSIC.HTML
• For public domain American popular music see
  – http://levysheetmusic.mse.jhu.edu
Breaking up Long Methods

• Music often has verses and a refrain
  – You play a verse and then the refrain
  – And then play the next verse and then the refrain

• You could put all of this in one big method
  – Put it would be more work and harder to change

• A better approach is to break the playing of a song into several methods
  – And create one method that calls the others
public void playJingleBells()
{
    // play verse 1
    playJingleBellsV1();

    // play refrain
    playJingleBellsRefrain();

    // play verse 2
    playJingleBellsV2();

    // play refrain
    playJingleBellsRefrain();
}
Private Methods

• If you want code in another class to be able to invoke a method in your class
  – Make the visibility public

• If you don’t want code in another class to be able to invoke a method in your class
  – Make the visibility private

• Private methods can only be invoked by code in the same class
  – So they can be changed without worrying about affecting another class
Summary

• Class fields
  – Are allocated space in the object that defines the class (an object of the class `Class`)
  – Each object has a reference to the object that defines a class and can access Class fields
  – Are declared using the static keyword

• Constants
  – Don’t change
  – Are declared using the final keyword

• Private Methods
  – Can only be invoked by code in the same class
Today -- new Stuff!

- Two kinds of methods:
  - object
  - class

- Creating Classes
  - identifying objects and classes
  - constructors
  - adding a method, accessors and modifiers, creating a main method
  - comments, javadocs
Identifying Objects and Classes

- Object-oriented programs
  - Consist of interacting objects
    - Which are defined by and created by classes
- To identify the objects in a task
  - What are the things that are doing the work or being acted upon?
  - How do you classify them?
  - What data (fields) do they need to know to do the task?
  - What procedures (methods) do they need?
Identifying the Objects and Classes

- Say that we want to write a program to do a slide show
  - A series of pictures shown one after the other with some time waiting between the pictures
- One way to start is to underline the nouns
  - Slide show, picture, wait time
- A slide show has pictures and a time to wait between pictures
Class Definition

• Each class is defined in a file
  – With the same name as the class: SlideShow.java

• Class names
  – Are singular (SlideShow not SlideShows)
  – Start with an Uppercase letter
  – The rest of the word is lowercase
  – upperCase the first letter of each additional word

• The syntax for a class definition is:
  – visibility class Name {}
Class Declaration

- To declare a SlideShow class
  - Click on the New button in DrJava
- Type in:
  ```java
  public class SlideShow
  {
  }
  ```
- Save it in SlideShow.java
  - Click on File then Save
- Click the Compile All button to compile it

based on slides by Barb Ericson,
Georgia Institute of Technology
SlideShow Fields

• A SlideShow has pictures and a wait time
  – What type should we use for each of these?
    • For the pictures we can use a 1-D array
    • For wait time we can use integer to hold the number of milliseconds to wait
    • Use Thread.sleep(waitTime) to wait for waitTime number of milliseconds
    • This can cause an exception so write the method to throw Exception by adding throw Exception to the method definition line
Declaring Fields

- **Syntax**
  - visibility type name;
  - visibility type name = expression;
- **Usually use private for the visibility**
  - So that other classes can’t access it directly
- **The type is any of the primitive types, a class name, or an interface name**
- **Arrays are declared with [] after the type or after the name**
  - type[] name; or type name[];
- **Names start with a lowercase letter**
  - The first letter of each additional word is upperCased
Default Field Values

• If you don’t specify an initial value for a field
  – It will get one anyway when it is created
    • Numbers = 0
    • Objects = null (not referring to any object yet)
    • boolean = false

```java
public class SlideShow {

    private Picture[] pictureArray;
    private int waitTime = 2000;
}
```

Initial value will be null

---

based on slides by Barb Ericson,
Georgia Institute of Technology
Testing the SlideShow Class

• Add the fields to the class definition and compile it
• Try the following in the interactions pane
  SlideShow slideShowObj = new SlideShow();
  System.out.println(slideShowObj);
• What happens?
  – SlideShow@2bd3a <<< you may not get the exact same thing
What Happened? (Inherited Methods)

• When you executed
  – System.out.println(slideShowObj);

• The class SlideShow was checked for a toString method
  – Since it didn’t have one the parent class was checked for a toString method
    • The one in Object was executed
      – Which prints the hash code for the object

• The SlideShow class inherited the toString method from the Object class
How Inheritance Works

• When a method is invoked on an object
• We first check for that method in the object that defines the object’s class
• If it isn’t there we look in the parent of that class
All Classes Inherit from Object

- If you don’t specify the parent class when you declare a class
  - The class with inherit from java.lang.Object
- You can specify the parent class
  - Add extends Parent to the class declaration
    
    public class SlideShow extends Object

- A declaration of
  
  public class SlideShow

- Is the same as
  
  public class SlideShow extends Object

based on slides by Barb Ericson,
Georgia Institute of Technology
Getting the Class

• An object keeps a reference to the class that created it
  – You can get this class with
    • Class currClass = obj.getClass();

• Each class keeps a reference to its parent class
  – You can get this class with
    • Class parentClass = currClass.getSuperclass();

• Try the following:
  SlideShow showObj = new SlideShow();
  Class showClass = showObj.getClass();
  System.out.println(showClass);
  Class parentClass = showClass.getSuperclass();
  System.out.println(parentClass);
Overriding an Inherited Method

• If a class defines a method with the same name, parameter list, and return type as an inherited method
  – This method will be called instead of the parent method

• To override Object’s toString add this one to SlideShow:

```java
public String toString()
{
    return "A slide show with " +
    this.pictureArray.length + " pictures and " +
    "a wait time of " + this.waitTime;
}
```
Testing toString

- Compile SlideShow.java
- Type the following in the interactions pane
  ```java
  SlideShow showObj = new SlideShow();
  System.out.println(showObj);
  ```
- What do you get this time?
  - And why?
- Can you fix this?
Summary

• Object-oriented programs
  – Have interacting objects

• To decide what classes to create
  – Identify the objects doing the action or being acted upon
    • And classify them (what type of thing are they?)

• To declare a class
  – public class SlideShow{}

• To declare a field
  – private type fieldName;

• All classes inherit from Object
  – Inherit the toString() method

• Add a toString() method to your own classes
  – To override the inherited method
Today -- new Stuff!

• Two kinds of methods:
  – object
  – class

• Creating Classes
  – identifying objects and classes
  – constructors
  – adding a method, accessors and modifiers, creating a main method
  – comments, javadocs
Constructors

- Are used to initialize the fields of an object
  - To other than the default values or assigned values
- You can have more than one constructor
  - As long as the parameter lists are different
  - This is called overloading constructors
- Syntax
  - visibility ClassName(paramList) {}
- Example
  public SlideShow(Picture[] pictArray)
  {
    this.pictureArray = pictArray;
  }

based on slides by Barb Ericson,
Georgia Institute of Technology
Creating 1D Arrays

- You can declare an array using
  - `Type[] arrayName;`
- You can create an array using
  - `new Type[size];`
- You can declare an array and create it at the same time
  - `Type[] arrayName = new Type[size];`
- Array indices start at 0 and end at length – 1
- You can get the length of an array using
  - `arrayName.length`
- You can add an element to an array using
  - `name[index] = Object;`
Add a Constructor

• Add the following after the field declarations to SlideShow.java:
  
  ```java
  public SlideShow(Picture[] pictArray)
  {
      this.pictureArray = pictArray;
  }
  ```

• Compile and test

  ```java
  SlideShow showObj = new SlideShow();
  ```
Why did you get an Error?

• We hadn’t declared any constructors before we added this one
  – But a constructor is called each time a new object is created
  – We didn’t provide one so the compiler added a no-argument constructor
    • One that takes no parameters and leaves the fields with their default or assigned values

• But once you add a constructor
  – The compiler will not add any for you
    • So now you get an error when you try to use a no-argument constructor
Adding a No-Argument Constructor

- Add the following constructor to the Student class
  
  ```java
  public SlideShow() {}
  ```

- Now test it again with:
  
  ```java
  SlideShow showObj = new SlideShow();
  System.out.println(showObj);
  ```

- Also try:
  
  ```java
  Picture[] pictArray = new Picture[5];
pictArray[0] = new Picture(FileChooser.getMediaPath("beach.jpg"));
pictArray[1] = new Picture(FileChooser.getMediaPath("blueShrub.jpg"));
pictArray[2] = new Picture(FileChooser.getMediaPath("church.jpg"));
pictArray[3] = new Picture(FileChooser.getMediaPath("eiffel.jpg"));
pictArray[4] = new Picture(FileChooser.getMediaPath("greece.jpg"));
SlideShow vacShow = new SlideShow(pictArray);
System.out.println(vacShow);
  ```
Tracing Execution

• One way to trace what is happening in your program is
  – To add `System.out.println()` statements

• Add these to print out the value of the picture array both before and after it is set
  – `System.out.println(this.pictureArray);`
  – `this.pictureArray = pictArray;`
  – `System.out.println(this.pictureArray);`
Debuggers

- You can use a debugger to find the cause of bugs (errors in your program)
  - A moth caused one bug
  - http://www.jamesshuggins.com/h/tek1/first_computer_bug.htm
- And to trace execution to see what is happening
  - Which constructor is executed or what method is executed
  - What values are in the fields
DrJava’s Debugger

- Click on Debugger in the menu
  - Then check the Debug Mode checkbox

Based on slides by Barb Ericson,
Georgia Institute of Technology
Setting a Breakpoint

• When you use a debugger you often want to set places to stop execution
  – Each place to stop at is a breakpoint

• Once execution has stopped there
  – You can check the value of parameters and fields

• To set a breakpoint
  – Right click on a line of code
  – Pick “Toggle Breakpoint”
  – It will be highlighted in red

based on slides by Barb Ericson, Georgia Institute of Technology
Showing a Breakpoint

• Lines with breakpoints are highlighted in red in DrJava
• Set a breakpoint at the line that sets the picture array
Testing a Breakpoint

• Try the constructor again that takes an array of pictures

• Execution should stop at the breakpoint
  – And the color will change to blue
Checking Values

• Execution stops before the breakpoint line is executed
  – So the array hasn't been set yet
  – Check this by printing out the value of it in the interactions pane
    • this.pictureArray
  – Then click on the Step Over button
    • To let the current line of code be executed
  – And check the values again
Debugging Options

• **Step Over**
  – Execute the current line of code and then stop again before you execute the next line of code

• **Step Into**
  – If the line of code that we are stopped at has a method call in it stop at the first line in the called method

• **Resume**
  – Continue execution at the current point
    • Until the next breakpoint
    • Or the program ends

• **Step Out**
  – Execute the rest of the current method and stop at the first line after the call to this method

• **You can quit debugging by clicking on the X**

---

based on slides by Barb Ericson, Georgia Institute of Technology
Adding a Constructor Exercise

• Create another constructor in the SlideShow class
  – That takes both the array of pictures and the time to wait between pictures
    public SlideShow(Picture[] pictArray, int time)
  – Use the debugger to check what happens during execution of this constructor
Summary

• Constructors initialize the fields in an object
• To declare a constructor
  – public Class_Name(paramList) {}
    • No return type
    • Same name as the class
• You can overload constructors
  – The parameter lists must be different
• Use a debugger
  – To watch what happens during execution
Today -- new Stuff!

• Two kinds of methods:
  – object
  – class

• Creating Classes
  – identifying objects and classes
  – constructors
  – adding a method, accessors and modifiers, creating a main method
  – comments, javadocs
Showing the Slide Show

• Now that a slide show has an array of slides we would like to
  – Show the pictures in the array
    • We can loop through the elements of the array
      – And show the current picture
      – And wait for the wait time
      – Then hide the current picture
    • We need to be careful of
      – A null pictureArray
Create a Method Exercise

• Create a method `show` that will first check that the picture array isn't null
  – And if it isn't will loop through the pictures in the array
    • Showing the current picture
    • Waiting till the wait time has passed
    • Hiding the current picture
Accessing Fields from Other Classes

• Fields are usually declared to be private
  – So that code in other classes can’t directly access and change the data

• Try this in the interactions pane
  – System.out.println(showObj.pictureArray);

• You will get an exception
  – Short for exceptional event – error

• Outside classes can not use object.field to access the field value
  – Unless it is declared with public visibility
Accessors and Modifiers

• Accessors
  – Are public methods that return data
    • In such a way as to protect the data for this object
    • Syntax
      public fieldType getFieldName()
    • Example
      public String getName() { return this.name; }

• Modifiers
  – Are public methods that modify data
    • In such a way as to protect the data for this object
    • Syntax
      public returnType setFieldName(type name);
    • Example
      public void setName(String theName)
      {this.name = theName; }
Naming Conventions

• Accessors – also called Getters
  – Use getFieldName for non boolean fields
  – Use isFieldName for boolean fields

• Modifiers – also called Setters and Mutators
  – Use setFieldName
  – Sometimes return a boolean value to indicate if the value was set successfully

• Examples
  – getName and setName
Creating SlideShow Accessors

- Add a method to get the wait time
  ```java
  public int getWaitTime()
  ```

- What about a method to get the array of pictures?
  - If someone gets the array s/he can directly change the pictures in the array
  - It is safer to return the picture at an index
  - Then other classes can’t directly change the array of pictures
Exercise

• Create a method that returns the wait time
• Create a method that returns the picture at a given index in the array
  – If the array is null return null
  – If the index isn't valid return null
Creating Slide Show Modifiers

• We need public methods
  – That let other classes set the time to wait between pictures
  – Our class is responsible for making sure this only happens in such a way
    • as to keep the data valid and not cause errors

• Setting the wait time
  – The wait time must be > 0

• Setting an array of pictures
  – We can decide if this can be changed or not when it isn’t null
Set Picture Array Modifier

- Setting the array of pictures only if it is currently null

```java
public boolean setPictureArray(Picture[] theArray)
{
    boolean result = false;
    if (this.pictureArray == null)
    {
        this.pictureArray = theArray;
        result = true;
    }
    return result;
}
```
public void setWaitTime(int time) {
    // check that it is a valid wait time
    if (time >= 0)
        this.waitTime = time;
}
Add a Field Exercise

• Add a title field to the SlideShow class
• Add an accessor to get the value of this field
• Add a modifier to set the value of this field
• Modify the show method to first create a blank picture with the title on it and show that as the first picture in the slide show
Adding a Main Method

- We have been typing stuff in the interactions pane in DrJava
  - To try out Java code and to try methods
- Most development environments make you write a main method to start execution
  - DrJava allows this too

Each class can have a main method declared as follows:

- public static void main(String[] args)
  - It is public so that it can be called by other classes
  - It is static because no object of the class exists when it is executed
  - It doesn’t return anything so the return type is void
  - You can pass several arguments to the main method and these are put in an array of strings
Main Method

- Add a main method to SlideShow
  - Put the statements that you have been doing in the interactions pane in the main method

```java
public static void main(String[] args) throws Exception {
    Picture[] pictArray = new Picture[5];
pictArray[0] = new Picture(FileChooser.getMediaPath("beach.jpg"));
pictArray[1] = new Picture(FileChooser.getMediaPath("blueShrub.jpg"));
pictArray[2] = new Picture(FileChooser.getMediaPath("church.jpg"));
pictArray[3] = new Picture(FileChooser.getMediaPath("eiffel.jpg"));
pictArray[4] = new Picture(FileChooser.getMediaPath("greece.jpg"));
    SlideShow vacShow = new SlideShow(pictArray);
    vacShow.show();
}
```
Execute the Main Method

• In DrJava you can run the main method in the class that is displayed in the definitions pane
  – By clicking on Tools then Run Document’s Main Method (or press key F2)

• It will do
  – java SlideShow
  – In the interactions pane
  – Which executes the main in the SlideShow class
Summary

• Classes have fields, constructors, and methods
• Constructors are used to initialize fields in the object
• Fields are usually declared to be private
  – To protect the data from misuse by other classes
  – So you need to provide public accessor (getter) and modifier (setter) methods
    • That still protect the data
• Use a main method to begin execution
  – public static void main(String[] args) {}
Today -- new Stuff!

• Two kinds of methods:
  – object
  – class

• Creating Classes
  – identifying objects and classes
  – constructors
  – adding a method, accessors and modifiers, creating a main method
  – comments, javadocs
Comments

• You should add comments to your code
  – To make it easier to read and change
• Comments are ignored by the compiler
  – Not added to the byte codes
• Java has 3 kinds of comments
  – // comment ends at the end of this line
  – /* comment ends with next */
  – /** Javadoc comment that ends with */
    • can be used by the javadoc utility to create HTML documentation
Javadoc Comments

- Add a comment before the class definition
  - That explains the purpose of this class
  - And says who wrote it
    - @author Barb Ericson

/**
 * Class that represents a slide show. A slide show has
 * an array of pictures, a time to wait between pictures,
 * and a title that is shown at the beginning of the show.
 *
 * @author Barb Ericson
 */

public class SlideShow
Method Comments

- Add a comment before each method
- What the parameters are
  - @param name info
- What is returned
  - @return info

```java
/**
 * Method to change the time to wait
 * between pictures
 * @param time the new time to use
 * in milliseconds
 */
public void setWaitTime(int time)
{
    if (time >= 0)
        this.waitTime = time;
}
```
Previewing Javadoc HTML

- Click on Tools
- Click on Preview Javadoc for Current Document
  - This will generate the HTML from the javadoc comments and display it
- The HTML document will display
Generating all HTML for Directory

• In DrJava click on the Javadoc button
  – to create the HTML documentation
  – based on the Javadoc comments
• This will generate HTML for all files in the same directory as all open files
• Generates an index.html as a starting point
Javadoc Exercise

• Add a class javadoc comment and method javadoc comments to the SlideShow class
• Execute Javadoc and check out the created documentation
Summary

• Comments are added to make a program
  – Easier to read and understand
  – Comments are ignored by the compiler

• There are three types of comments in Java
  – // end of line
  – /* multi line */
  – /** java doc */

• Javadoc is a utility that comes with the jdk
  – Produces HTML documentation from Javadoc comments