Creating Classes part 1

Barb Ericson
Oct 2010

Learning Goals

- Define a class
  - Including fields, constructors, and methods
- Override an inherited method
- Start using a debugger
- Overload constructors
- Create, initialize, access, and process an array
- Create accessor and modifier methods
- Introduce runtime exceptions
- Create a main method
- Create Javadoc comments
- Introduce dynamic binding

Classes in Alice

- The only way to create a class in Alice 2.2 is to modify an existing class
  - And save it out with a new class name
  - Like the CleverSkater
- Each class in Alice has a set of properties and a set of methods
  - In Java we call properties fields
  - In Java we also have a special type of method that initializes the newly created object
    - Called constructors

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

UML Diagram for SlideShow

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* {}
- Inside the class definition goes:
  - Fields, constructors, and methods

Class Declaration

- To declare a SlideShow class
  - Click on the New button in DrJava
- Type in:

```
public class SlideShow
{
}
```

- Save it in SlideShow.java
  - Click on File then Save
- Click the Compile All button to compile it

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
      - 1000 milliseconds is one second
    - This can cause an exception so write the method to throw Exception by adding throw Exception

Declaring Fields

- Syntax
  - visiblity 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

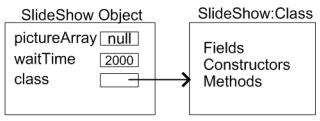
- If you don't specify an initial value for a field
  - It will get one anyway when it is created
    - Numbers = 0
    - Objects = null (not referring to any object yet)
    - boolean = false

```
public class SlideShow
{
  ////////////// fields ////////////////////////////////////////
  private int waitTime = 2000;
  private Picture[] pictureArray;
}
```

Initial value will be null

All objects know what type they are

- Objects keep a reference to the class that created them
  - All methods are first looked for first in this class



Testing the SlideShow Class

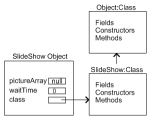
- Add the fields to the class definition and compile it
- Try the following in the interactions pane
  - SlideShow slideShowObj = new SlideShow();
  - System.out.println(slideShowObj);
  - SlideShow show2 = new SlideShow();
  - System.out.println(show2);
- What happens?

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 hash code is a hex number that is unique 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

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:

```
public String toString()
{
   return "SlideShow object with a wait time of " + this.waitTime;
}
```

Testing toString

- Compile SlideShow.java
- Type the following in the interactions pane

```
SlideShow showObj = new SlideShow();
System.out.println(showObj);
```

- What do you get this time?
  - And why?

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(int theTime)
{
  this.waitTime = theTime;
}
```


Trying the new constructor

> System.out.println(new SlideShow(5000));
SlideShow object with a wait time of: 5000

Try the no-argument constructor as well:
> System.out.println(new SlideShow());
What happens?

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

Add a no-argument constructor

```
////////////// constructors //////////
public SlideShow() {}

public SlideShow(int theTime)
{
  this.waitTime = theTime;
}
```


Adding a No-Argument Constructor

- Add the following constructor to the SlideShow class
  - public SlideShow() {}
- Now test it again with:

```
SlideShow showObj = new SlideShow();
System.out.println(showObj);
```

- Also try:

```
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 in the constructor to print out the value of the wait time both before and after it is set

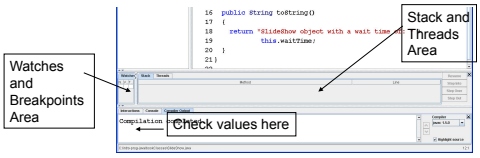
```
System.out.println(this.waitTime
this.waitTime = theTime;
System.out.println(this.waitTime);
```


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

- You can turn on the debugger in DrJava
  - Click on Debugger and then check Debug Mode
    - DrJava will add new windows to the bottom of the window

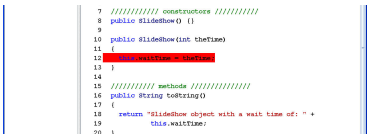


Setting a Breakpoint

- 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

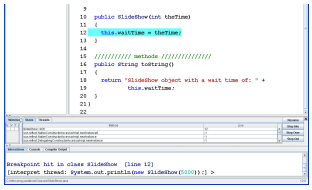
Showing a Breakpoint

- 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 integer wait time
- 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.waitTime;
    - > 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

Adding a Constructor Exercise

- Create another constructor in the SlideShow class
  - One that takes the array of pictures

```
public SlideShow(Picture[] pictArray)
{                                    }
  this.pictureArray = pictArray;
}
```

- We need to create an array of pictures to pass in to this constructor

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];
- You can add an element to an array using
  - name[index] = Object;
- You can initialize the contents of an array when you create it
  - *type*[] *name* = {*elem1*,*elem2*,*elem3*,...};

Creating an array of pictures

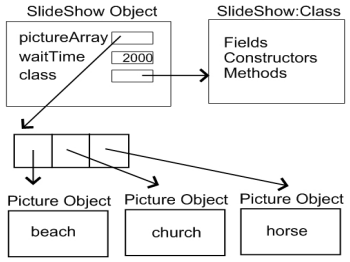
```
> Picture pict1 =
    new Picture(FileChooser.getMediaPath("beach.jpg"));
> Picture pict2 =
    new Picture(FileChooser.getMediaPath("church.jpp"));
> Picture pict3 =
    new Picture(FileChooser.getMediaPath("horse.jpg"));
> Picture[] pictArray = {pict1, pict2, pict3};
```


Another way to set-up the array

- You can also create the array first and then fill in the references to the pictures

```
> Picture[] pictArray = new Picture[3];
> pictArray[0] =
    new Picture(FileChooser.getMediaPath("beach.jpg"));
> pictArray[1] =
    new Picture(FileChooser.getMediaPath("church.jpg"));
> pictArray[2] =
    new Picture(FileChooser.getMediaPath("horse.jpg"));
> System.out.println(new SlideShow(pictArray);
SlideShow object with a wait time of: 2000
```

What does this look like?



Adding a getNumPicts() method

- We might want to also print out the number of pictures in the slide show
  - This is something other classes might want to know
  - So let's add it as a public method

```
public int getNumPicts()
{
  // if no picture array then there are no pictures
  if (this.pictureArray == null)
  {
    return 0;
  }
  // else return the number of pictures in the
    array
  else
  {
    return pictureArray.length;
  }
}
```


Modifying toString

- Now we can modify toString to tell us how many pictures are in the slide show

```
public String toString()
{
  return "SlideShow object with a wait time of: " +
        this.waitTime + " and " +
        this.getNumPicts() + " pictures";
}
```


Challenge

- Set a breakpoint in the toString method and use the Step Into to follow the call to getNumPicts
  - Step into will take you into the called method
  - Step over will execute the method and stop before the next line

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

Thread.wait

- Use Thread.sleep(waitTime) to wait for waitTime number of milliseconds
  - 1000 milliseconds is one second
- This can cause an exception
  - exceptional event - like if someone hits the reset button in DrJava while we are waiting
  - write the method to throw Exception by adding throws Exception

    public void show() throws Exception

The show method

Add this method to the SlideShow class

```
public void show() throws Exception
 {
   for (Picture pictObj : this.pictureArray)
   {
     pictObj.show();
     Thread.sleep(this.waitTime);
     pictObj.hide();
   }
 }
```


Testing SlideShow

```
> Picture pict1 =
   new Picture(FileChooser.getMediaPath("beach.jpg"));
> Picture pict2 =
   new Picture(FileChooser.getMediaPath("church.jpp"));
> Picture[] pictArray = {pict1, pict2};
> SlideShow show1 = new SlideShow(pictArray);
> show1.show();
```


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 or Mutators
  - 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
  public int getWaitTime() { **return this**.waitTime; }
- 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 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 change the fields
  - Our class is responsible for making sure this only happens in such a way
    - as to keep the data valid and not cause errors
- Changing a picture in the slide show
  - The picture array can't be null and the picture can't be null
- Setting the picture array
  - Only if it is currently null
- Setting the wait time
  - The wait time must be > 0

setPicture method

```
public boolean setPicture(int index, Picture pict)
{
  if (pict == null || this.pictureArray == null)
    return false;
  else
  {
    this.pictureArray[index] = pict;
    return true;
  }
}
```


Challenge

- Is there anything else that setPicure should be checking to make sure that the data is valid?

Set Picture Array Modifier

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

```
public boolean setPictureArray(Picture[] theArray)
 {
   boolean result = false;

   if (this.pictureArray == null)
   {
     this.pictureArray = theArray;
     result = true;
   }
   return result;
 }
```


Wait Time Modifier

```
public boolean setWaitTime(int theTime)
 {
   boolean result = false;
   if (theTime >= 0)
   {
     this.waitTime = theTime;
     result = true;
   }
   return result;
 }
```

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

```
public static void main(String[] args) throws Exception
 {
   Picture p1 =
    new Picture(FileChooser.getMediaPath("beach.jpg"));
   Picture p2 =
    new Picture(FileChooser.getMediaPath("church.jpg"));
   Picture[] pictArray = {p1,p2};
   SlideShow show1 = new SlideShow(pictArray);
   System.out.println(show1);
   show1.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

Comments

- You should add comments to your code
  - To make it easier to read and change
- Comments are ignored by the complier
  - Not added to the byte codes
- Java has 3 kinds of comments
  - // comment ends at the end of this line
  - /* multi-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 defines a slide show.  A slide show
 * has pictures and a time to wait between showing the
 * pictures
 * @author Barb Ericson
 */
public class SlideShow
```

Multiple Authors

- Simply add a @author tag for each author

```
/**
 * 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 Mark Guzdial
 * @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

```
/**
 * Method to set a picture at the passed
 * index
 * @param index which one to change
 * @param thePict the picture to use
 * @return true if success else return false
 */
public boolean setPicture(int index,
                          Picture thePict)
```


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 HTML for a 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 Javadoc comments to the SlideShow and Student classes
  - Add a class comment with @author tag
  - Add method comments
  - Add comments to the constructors
- Execute Javadoc and check out the created documentation

Creating an Inherited Class

- Let's create a class ConfusedTurtle that inherits from the Turtle class
  - But when a ConfusedTurtle object is asked to turn right it will turn left and vice versa
- To inherit from another class
  - Add extends ClassName to the class declaration
    - To call a method in a parent class use super.method(arguments);

ConfusedTurtle class

```
public class ConfusedTurtle extends Turtle
{
 /**
  * Method to turn right (but a confused
  * turtle will actually turn left)
  */
 public void turnRight()
 {
  super.turnLeft();
 }
}
```


ConfusedTurtle class - cont

```
 /**
  * Method to turn left (but a confused
  * turtle will actually turn right)
  */
 public void turnLeft()
 {
  super.turnRight();
 }
}
```


Compile Error?

- If you try to compile ConfusedTurtle you will get a compiler error
  - Error: cannot resolve symbol
  - symbol: constructor Turtle()
  - location: class Turtle
- Why do you get this error?

Inherited Constructors

- When one class inherits from another all constructors in the child class will have an implicit call to the no-argument parent constructor as the first line of code in the child constructor
  - Unless an explicit call to a parent constructor is the first line of code in the constructor

```
super(argumentList);
```


Why is an Implicit Call to Super Added?

- Object fields are inherited from a parent class
  - But object fields should be declared private
    - Not public, protected, or package visibility
      - Lose control over field at the class level then
  - But then subclasses can't directly access inherited object fields
  - How do you initialize inherited fields?
    - By calling the parent constructor that initializes them
      - Using super(paramList);

Explanation of the Compile Error

- There are no constructors in ConfusedTurtle
  - So a no-argument one is added for you
    - With a call to super();
  - But, the Turtle class doesn't have a no -argument constructor
    - All constructors take a world to put the turtle in
- So we need to add a constructor to ConfusedTurtle
  - That takes a world to add the turtle to
    - And call super(theWorld);

Add a Constructor that takes a World

```
/**
 * A constructor that takes a ModelDisplay object
 * @param modelDisplayObj the thing that does the
   display
 */
public ConfusedTurtle(ModelDisplay modelDisplayObj)
{
  // use parent constructor
  super(modelDisplayObj);
}
```


Try out ConfusedTurtle

```
> World world = new World();
> ConfusedTurtle fred = new ConfusedTurtl
  e(world);
> fred.forward();
> fred.turnLeft();
> fred.forward();
> fred.turnRight();
> fred.forward();
```


Method Resolution

- You can set the value of an object reference to be of the declared type

  ConfusedTurtle fred = new ConfusedTurtle(world);

  – Or any subclass of the declared type

  Turtle fred = new ConfusedTurtle(world);
- Methods are executed at run-time
  - Based on the actual type of the object
    - The class that created it
      - So the turtle fred will act like a confused turtle

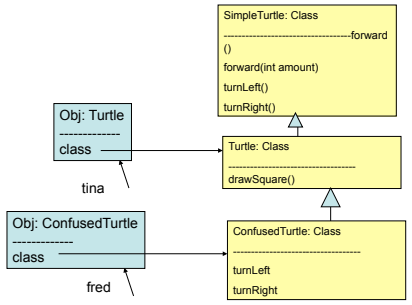
Override Methods

- Children classes inherit parent object methods
  - The confused turtle knows how to go forward
    - Inherited from Turtle which inherits from SimpleTurtle
- Children can override parent object methods
  - Have a method with the same name and parameter list as a parent method
    - This method will be called instead of the parent method
      - Like turnLeft or turnRight

What is Happening?

- Each time an object is asked to execute a method
  - Check the class that created the object to see if the method is defined in that class
    - If it is, it will execute that method
    - If it isn't, next check the parent class of the class that created it
      - that method will execute if one is found
      - If no method with that name and parameter list is found, check that classes parent
        - » Keep going until you find the method

Method Overloading



Exercise

- Create a StubbornTurtle class
  - That has a 50% chance of doing what you ask
  - You can use Math.random() to get back a number from 0 to not quite 1 (not inclusive)
  - You can check if the random number is greater than .5 and if so call the parent method to do the action

Adding Fields and Methods to a Subclass

- What if we want to play music while the slide show is playing?
  - Create a MusicalSlideShow that inherits from SlideShow
- Add a field for a sound clip (using the Sound class).
- We can override the show method to first start playing the sound and then call the parent's show method
- We can add methods to get and set the sound

Challenge

- What if the music is too long for the slide show?
- What if the music is too short for the slide show?
- Can you make the music match the length of the slide show?

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?)
- All classes inherit from Object
  - Inherit the toString() method
- Add a toString() method to your own classes
  - To override the inherited method
- You can create classes that specify the parent class
  - public class ConfusedTurtle extends Turtle
- Using a debugger can help you figure out what your program is doing