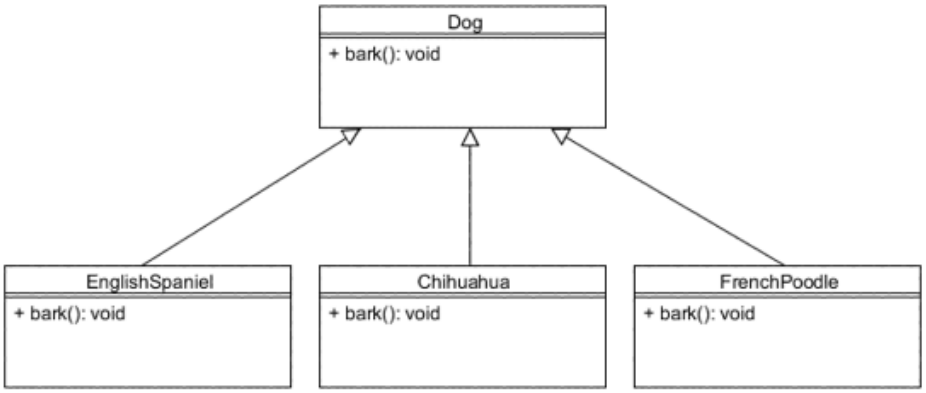
Polymorphism and Abstraction

Runtime Polymorphism

- basic definition
  - the ability to take on many shapes
- in programming
  - allows an object of **one** type be used as a reference to objects of other types and leave it up to the **compiler** to call the correct method for each of the different object types
    - the allows a single method call to behave differently, depending on the specific type of object it is associated with
  - You can create the overall OBJECT, then define the exact "behaviors" later!!!

Simple Polymorphism Example 1




```
public class Dog
{
	public void bark() {}

}

public class EnglishSpaniel extends Dog
{
	public void bark() { System.out.println( "\n I say old chap... I'm english"); }

}

public class Chihuahua extends Dog
{
	public void bark(){ System.out.println( "\n Yo quiero Taco Bell... I'm Spanish"); }
}

public class FrenchPoodle extends Dog
{
	public void bark(){ System.out.println( "\n Bonjour mon ami... I'm French"); }
}
```

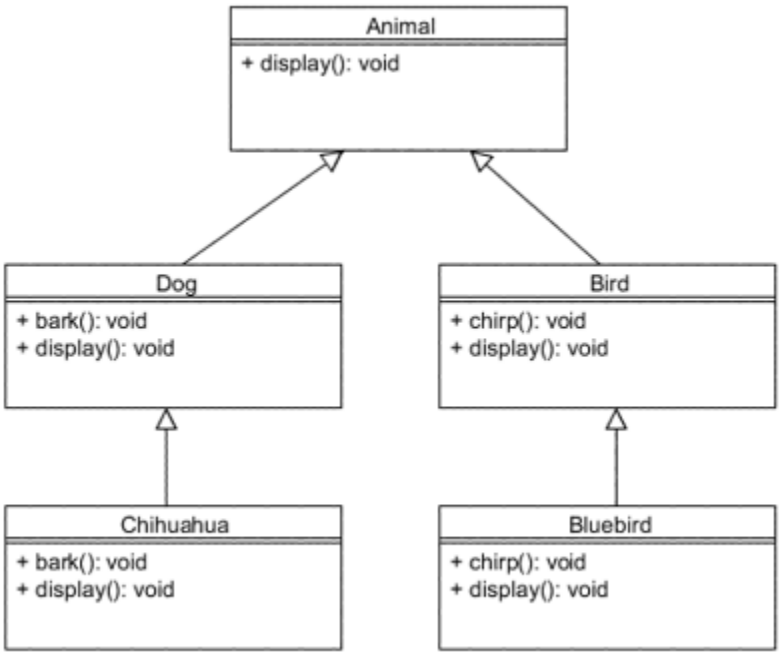
```
public class Driver
{

	public static void main(String [] args)
	{

		Dog [] dogCollection = {new Chihuahua(), new FrenchPoodle(), new EnglishSpaniel()};

		for (int i = 0; i < dogCollection.length; i++)
			dogCollection[i].bark();

		Dog Jason = new FrenchPoodle();
		Dog Killer = new Chihuahua();
	}

}
```

Problems | Javadoc | Declaration | Console

<terminated> PolymorphicDemo1 [Java Application] C:\Program Fi

```
Yo quiero Taco Bell... I'm Spanish

Bonjour mon ami... I'm French

I say old chap... I'm english
```

Simple Polymorphism Example 2




```
Animal anAnimal = new Chihuahua();
anAnimal.display(); // OKAY
anAnimal.bark();    // WRONG

Dog aDog = new Chihuahua();
aDog.bark();        // OKAY

Bird aBird = new Bluebird();
aBird.chirp();      // OKAY
```

// first line here is TOO generic

Defining the Exact Behaviors (later)

- this means we can create the Object (using the Super class) without determining which exact base object it is
- can have some decision about it later in the code WHILE the code is running!!

Example of Run-Time Polymorphism

Example 1

```
Dog Amy; // not fully instantiated

// MUCH later in the code

boolean flag = true;

if(flag == true) { Amy = new EnglishSpaniel(); } // finishing out the instance
else { Amy = new FrenchPoodle(); } // finishing out the instance


// Amy could be ONE of these two Dog's from here out!!
```

Example 2

```
Dog [] array; // an array of Dogs, just not sure which type yet

boolean flag = false;

if(flag == true)
{
	array = new EnglishSpaniel[5]; // finishing out the instance
	array[0] = new EnglishSpaniel(); // have to instantiate the object in each element
	array[1] = new EnglishSpaniel();
	array[2] = new EnglishSpaniel();
	array[3] = new EnglishSpaniel();
	array[4] = new EnglishSpaniel();
}
else
{
	array = new FrenchPoodle[3]; // finishing out the instance
	array[0] = new FrenchPoodle(); // have to instantiate the object in each element
	array[1] = new FrenchPoodle();
	array[2] = new FrenchPoodle();
}

for (int i = 0; i < array.length; i++) { array[i].bark(); }
```

Designing for Polymorphism

- remember, we are trying to save on coding
- but the design must be accurate to accomplish this and let Polymorphism do it work (then less work for us)

Lupoli's suggested object design

SuperClass

```
public class SuperClass
{

     // inherited members

     // constuctor to fill all inherited values

     // inherited methods

     // final methods

}
```

SubClasses

```
public class SubClass
{

     // un-inherited members, unique items to this subclass

     // constructor


     // un-inherited methods, unique methods to this subclass


     // overridden methods, more specific methods that the
     // superclass had, but need more IF the superclass does not have
// it finalized


     // class specific functions

}
```

Power of Polymorphism

- delay defining the complete object
  - since Plane and Boat are Vehicles, can create an incomplete instantiation
  - can complete later

The new BASE class

Vehicle Class

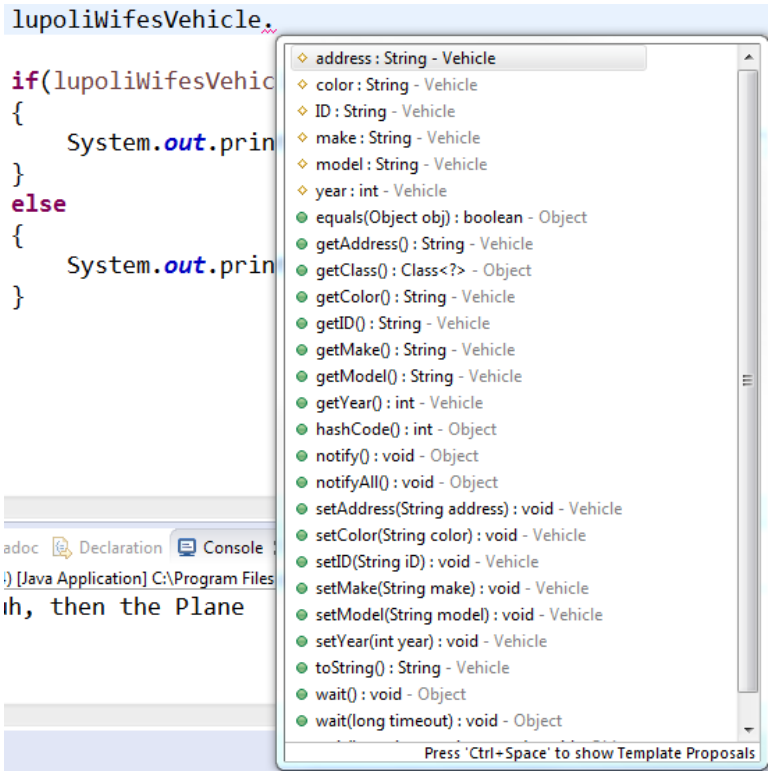
```
// overall vehicle details
float weight;
int yearMade; // would love to use short, but would have to cast
String make;
String model;
String color;
String ID;
int wheels;

// people related
int occupants;

// compartment related
int bathrooms;
int seats;
int doors;
int windows;

// engine related
String powerSource;
String engineType;
int horsePower;
int engineCount;
```

Delayed definition of an instance

```
// Plane and Boat were defined as extensions of Vehicle later


Vehicle lupoliBoat
    = new Boat(2015, "Viking", "Sports Cruiser", "MD1786237", "white", "800 Too Rich Rd.", "Driftwood");

Vehicle lupoliWifesVehicle;

boolean goodMood = true;


if(goodMood)
{
    lupoliWifesVehicle = new Plane(2015, "Cessena", "E180", "HG234878JHH", "red", "800 Too Rich Rd.", "N278234SD23");
}
else
{
    lupoliWifesVehicle = new Boat(2016, "Horizon", "E88", "MD189374", "white", "800 Too Rich Rd.", "Getaway");
}
```

- base class object recognition
  - even though a subclass, can determine subclass methods
  - notice, specific subclass methods/members are not present

base class recognition using Polymorphism



But no “registrationNumber”!!!

- subclass recognition using instanceOf
  - if you wish access to specific subclass
    - but need to cast after recognition

Subclass value recognition using "instanceof"

```
static void display(Vehicle x)
{

    if(x instanceof Plane)
    {
        Plane temp = (Plane) x; // since it is a plane, cast it to get full access
        temp.
    }
    else
    {
        Syste                                                    ');
    }

}

}
```

address : String - Vehicle
color : String - Vehicle
ID : String - Vehicle
make : String - Vehicle
model : String - Vehicle
year : int - Vehicle
equals(Object obj) : boolean - Object
getAddress() : String - Vehicle
getClass() : Class<?> - Object
getColor() : String - Vehicle
getID() : String - Vehicle
getMake() : String - Vehicle
getModel() : String - Vehicle
getRegistrationNumber() : String - Plane
getYear() : int - Vehicle
hashCode() : int - Object
notify() : void - Object
notifyAll() : void - Object
setAddress(String address) : void - Vehicle
setColor(String color) : void - Vehicle
setID(String iD) : void - Vehicle
setMake(String make) : void - Vehicle
setModel(String model) : void - Vehicle
Press 'Ctrl+Space' to show Template Proposals

lems @ Javadoc Declarat
ited> Driver (4) [Java Applicatio
mood huh, then t
5, 6:24:59 PM)
Writable

Polymorphism Rules

- the method must be defined as a method in the base class
- the subclass method must have the same “signature” as the base class method
- the visibility of the subclass method cannot be more restrictive
- the subclass method return type must be
    - the same as the base class method
    - or a type that is a subclass of the base class
- the derived class method call must be made using a variable of the base class type

Polymorphism using Arrays

- it only makes sense to use arrays to hold multiple instances that all are based off of some type
- while the creation of the array is nothing new, getting it truly set up and working is another
- need to make sure each ***individual*** item with the array full instantiated

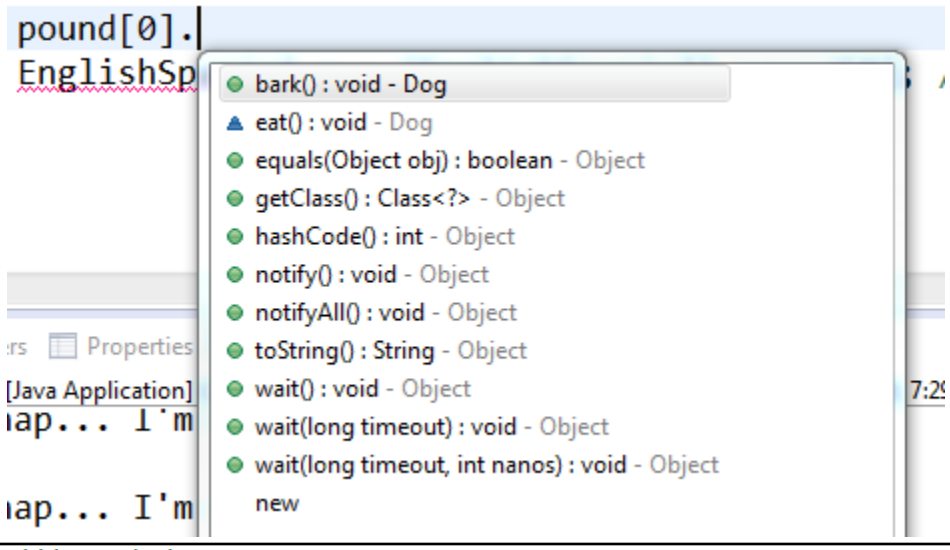
Polymorphism using Arrays

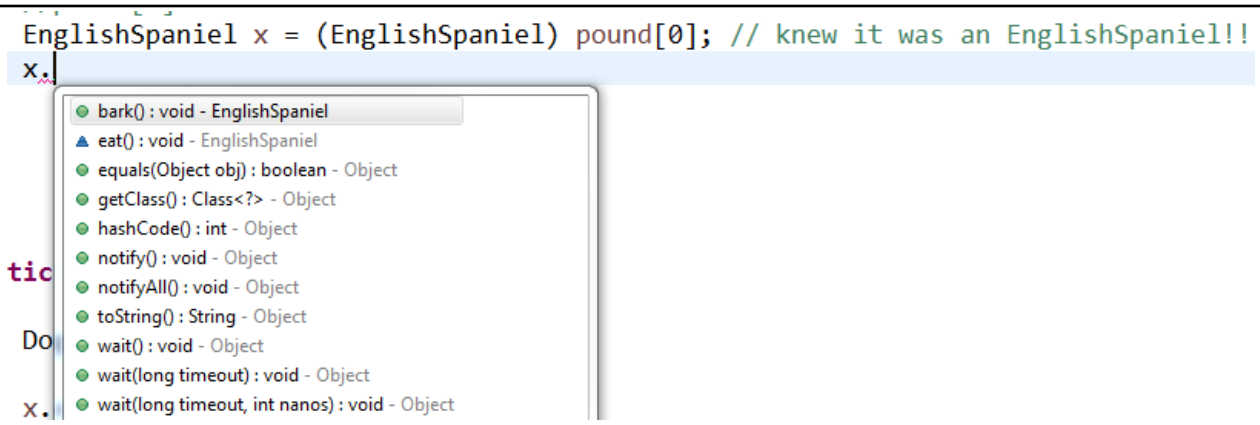
```
System.out.println("Trying an array of different Inheritance/Polymorphic instances");
Dog [] pound = new Dog[3];

pound[0] = new EnglishSpaniel();
pound[1] = new FrenchPoodle();
pound[2] = new Chihuahua();

displayArray(pound);
```

```
Trying an array of different Inheritance/Polymorphic instances
I eat fish and chips

 I say old chap... I'm english
I love stinky foot cheese!!

 Bonjour mon ami... I'm French
I love tomalies!!

 Yo quiero Taco Bell... I'm Spanish
```

- But when pulling from the array make sure to use the instanceof or cast it appropriately so that you can access the appropriate functions
  - casting is safer
- this will be easier later with Collections

Accessing an Array

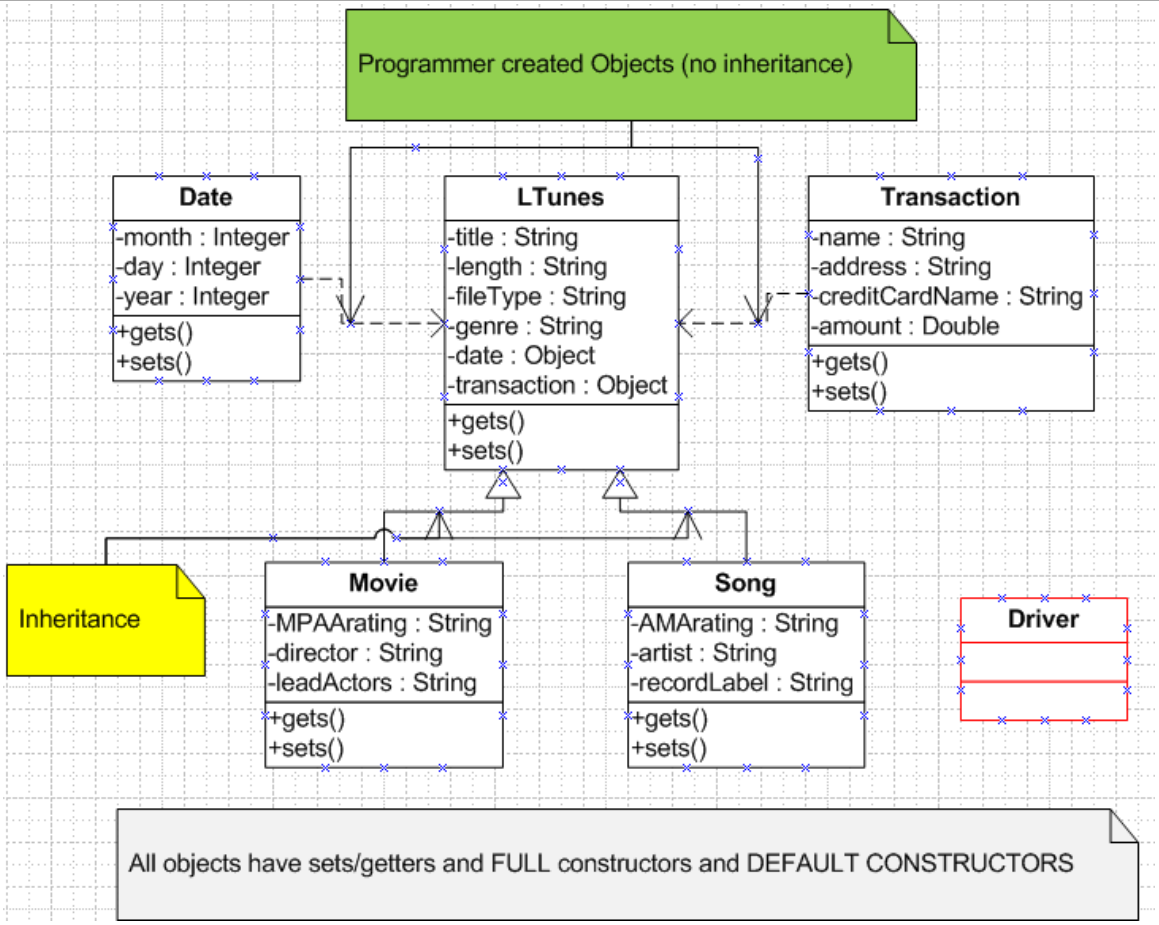




We need to talk about design for Abstraction

- UML Version vs. Coding

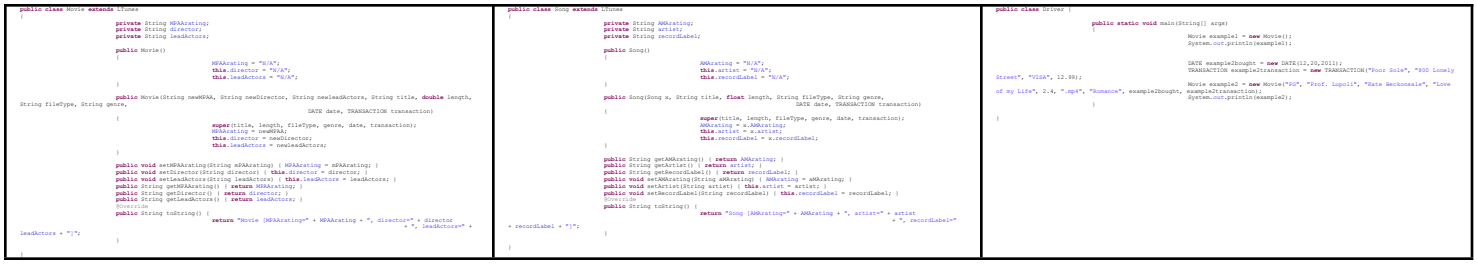
UML Version



1. What classes are listed?
2. What data members can you identify?
3. How can you tell if a class member (function or data) is public or private?
4. How can you tell if something is inherited?

Code (very small but matches above)

DATE.java	LTunes.java	Transaction.java
		
Move.java	Song.java	Driver.java



```
public class Movie extends LTunes
{
                    private String MPAArating;
                    private String director;
                    private String leadActors;

                    public Movie()
                    {
                                        MPAArating = "N/A";
                                        this.director = "N/A";
                                        this.leadActors = "N/A";

                    }

                    public Movie(String newMPAA, String newDirector, String newleadActors, String title, double length,
String fileType, String genre,
                                                                        DATE date, TRANSACTION transaction)
                    {
                                        super(title, length, fileType, genre, date, transaction);
                                        MPAArating = newMPAA;
                                        this.director = newDirector;
                                        this.leadActors = newleadActors;
                    }

                    public void setMPAArating(String mPAArating) { MPAArating = mPAArating; }
                    public void setDirector(String director) { this.director = director; }
                    public void setLeadActors(String leadActors) { this.leadActors = leadActors; }
                    public String getMPAArating() { return MPAArating; }
                    public String getDirector() { return director; }
                    public String getLeadActors() { return leadActors; }
                    @Override
                    public String toString() {
                                        return "Movie [MPAArating=" + MPAArating + ", director=" + director
                                                                                                                + ", leadActors=" +
leadActors + "]";
                    }

}
```

```
public class Song extends LTunes
{
                    private String AMArating;
                    private String artist;
                    private String recordLabel;

                    public Song()
                    {
                                        AMArating = "N/A";
                                        this.artist = "N/A";
                                        this.recordLabel = "N/A";

                    }

                    public Song(Song x, String title, float length, String fileType, String genre,
                                                                        DATE date, TRANSACTION transaction)
                    {
                                        super(title, length, fileType, genre, date, transaction);
                                        AMArating = x.AMArating;
                                        this.artist = x.artist;
                                        this.recordLabel = x.recordLabel;
                    }

                    public String getAMArating() { return AMArating; }
                    public String getArtist() { return artist; }
                    public String getRecordLabel() { return recordLabel; }
                    public void setAMArating(String aMArating) { AMArating = aMArating; }
                    public void setArtist(String artist) { this.artist = artist; }
                    public void setRecordLabel(String recordLabel) { this.recordLabel = recordLabel; }
                    @Override
                    public String toString() {
                                        return "Song [AMArating=" + AMArating + ", artist=" + artist
                                                                                                                + ", recordLabel="
+ recordLabel + "]";
                    }

}
```

```
public class Driver {

                    public static void main(String[] args)
                    {
                                        Movie example1 = new Movie();
                                        System.out.println(example1);

                                        DATE example2bought = new DATE(12,20,2011);
                                        TRANSACTION example2transaction = new TRANSACTION("Poor Sole", "800 Lonely
Street", "VISA", 12.99);
                                        Movie example2 = new Movie("PG", "Prof. Lupoli", "Kate Beckonsale", "Love
of my Life", 2.4, ".mp4", "Romance", example2bought, example2transaction);
                                        System.out.println(example2);
                    }

}
```

UML – Class Diagrams Options

- these are all the “standards” in diagram designs
  - certain symbols mean this….
  - http://www.objectmentor.com/resources/articles/umlClassDiagrams.pdf
  - http://www.agilemodeling.com/artifacts/classDiagram.htm
- these are all free options that can greatly help in design
  - http://www.smartdraw.com/resources/tutorials/uml-class-diagrams/
    - (nice and simple)
  - http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

Abstract ***and*** Implement as "Standards"

- intentionally staying away from the word "Template"
- using ***inheritance*** to force code completion
  - forcing the programmer to complete items, NOT the program
  - called a "contract" (since now you have to do it)
- think of it as setting a standard of what "should be" completed within the child/sub classes
  - so these (Abstract and Implement) would be ***base classes***
- while similar, they are not the same
  - differences shown below
- cannot ***directly*** instantiate either option
  - can create an instance of their subclass(es)

Class exercise for Abstract:

1. Create an abstract design for a Job
   a. what is every job have? (make them into data members)
2. As groups, create a specific instance (with a list of values) on paper
3. In that instance
   a. * those data members that came from the abstract base class
   b. $ those data members that are ***new*** to your specific class/instance

Abstract Class

- is meant to be extended (or meant to be a SUPER CLASS)
- is a generalization for more specific sub-classes
- but you cannot create an INSTANCE of this abstract class
  - Employee Lupoli = new Employee(…); // Nope
  - `Employee Lupoli = new SalariedEmployee(…);// Yup!! Polymorphism!!`
- holds abstract methods (and is required because of that method)
- can have
  - constructors, BUT, the sub-class must call it
    - since again, the abstract class cannot be directly instantiated
  - data members
  - public or protected methods
- cannot have
  - private ***methods***
    - private variables could be accessed by those public methods
- in Eclipse, when creating the class, make sure to select "abstract" in modifiers
- In UML, an abstract class has an italicized title (name)

Example Inheritance with Abstraction

```
public abstract class Employee
{
	protected String name;
	protected Date hireDate;

	public abstract double getMonthlyPay();
	// abstract since this method will be defined differently in the sub classes
	// both sub-class definitions need to return a double and be of the same name

	public Employee(String name, Date hireDate)
	{
		this.name = name;
		this.hireDate = hireDate;
	}

	public boolean samePay(Employee other)
	{
		return (this.getMonthlyPay() == other.getMonthlyPay());
	}

	// setters and getters

	public boolean equals(Object obj)
	{
		// …
	}
}
```

Hourly Employee

```
public class HourlyEmployee extends Employee
{
	private double wageRate;
	private double hours; //for the month

	public HourlyEmployee(String name, Date
hireDate, double wageRate,
			double hours)
	{
		super(name, hireDate);
		this.wageRate = wageRate;
		this.hours = hours;
	}

	// setters and getters

	public boolean equals(Object obj)
	{
		// …
	}

	public String toString()
	{
		// …
	}

}
```

Salaried Employee

```
public class SalariedEmployee extends Employee
{
	private double salary; //annual

	public SalariedEmployee(String name, Date
hireDate, double salary)
	{
		super(name, hireDate);
		this.salary = salary;
	}

	// setters and getters

	public boolean equals(Object obj)
	{
		// …
	}

	public String toString()
	{
		// …
	}

}
```

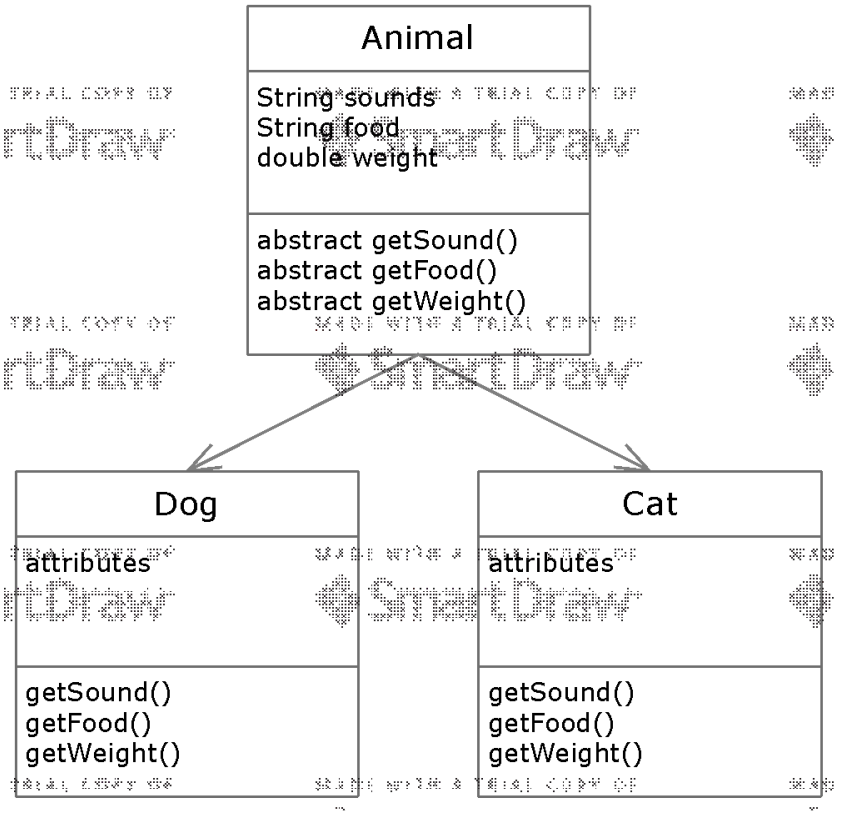
Driver

```
public class Driver
{

	public static void main(String[] args)
	{
		Employee Lupoli = new SalariedEmployee("Lupoli", new Date(7,1,2010), 80000);
		Employee Dima = new HourlyEmployee("Dima", new Date(8,18,2013), 50, 10);

		System.out.println(Lupoli);
		System.out.println(Dima);

	}

}
```

Abstract Methods

- An abstract method is like a placeholder for a method that will be fully defined in a descendent class(es)
  - o it postpones the definition and body of the method
  - o think of it as the implementation is being differed to be defined later
  - o could be used as a standard!!
    - in theory, abstract methods can set standards of what the sub-classes must later name and implement
- EVERY subclass should have these method(s)
  - ***o but the class itself must be abstract!!***
- has no method body (no {}s), and ends with a semicolon in place of its body.
- the methods ***cannot be*** private
- you can still call the SUB-CLASS version of the method to get information
- Bad Part
  - o if you have ONE abstract method, it must be inside an abstract class
- In UML, an abstract method has an italicized title (name)

Abstract – Setting the Standards



1. Download the code here and place together in a Project.
2. Finish out
    a. Create setters/getters functions in Employee
    b. the "toString" functions ***in the subclasses***
        i. (Hourly return wage_rate, hours, **name, and hireDate**)
        ii. (Salary return salary, **name, and hireDate**)

Use this to make above easier!!!

    c. finish/build the abstracted function ***in the sub-classes***
    d. don't worry about ANY of the other functions

Answer$_b$:

Interfaces

- ***completely*** abstract
  - all data members/methods are abstract
  - using public final static data members
- meant to be "implemented" (used by a sub-class)
  - cannot create a direct instance
- no constructor
  - this "class" is not meant to be instantiated, just implemented
  - sub-class that are implementing this Interface would have their own constructors
- concrete classes can implement many Interfaces
- will have unimplemented abstract methods
  - that the other sub-classes that implement must finish
- can use this as a standard or "contract", those that inherit MUST finish and define the job!
  - name of methods
  - name of variables
  - etc…
  - compiler will enforce
- notice no "class" (just interface) in the code

Interface (base) class

```
public interface Auctionable {  // notice "able" verb naming of class

     // data member
    public String condition = "New";

     // available conditions
    public static final int NEW          = 0;
    public static final int LIKE_NEW     = 1;
    public static final int REFURBISHED = 2;
    public static final int USED         = 3;

    // abstract methods to be implemented
    public String getDescription();
    public int getCondition();
}
```

Is the public keyword necessary within the class??

SubClass inheriting an Implementable class

```
public class Car implements Auctionable{
    private String make, model;
    public Car (String make, String model) {
        this.make = make;
        this.model = model;
    }
    public String getMake() { return make;  }
    public String getModel(){ return model; }
    public String toString(){
        return make + " " + model;
    }
    public String getDescription() {
        return "Low Mileage, New tires, AM/FM/CD";
    }
    public int getCondition(){
        return LIKE_NEW;
    }
}
```

1. On paper design a hierarchy of a Shape (base) with Rectangle, Circle, Equatorial Triangle
    a. what are the data members of each (not worried about functions yet)
    b. did any share data members (then should be in the base)
    c. did the specific shapes have any invariants?
2. While designing, did you design top-down (base then sub-classes) or bottom-up? (Signal your instructor when complete. Will need them for #3)
3. After ***assigned*** either an Abstract or Implements model, code the base and ONE specific shape (sub-class)
    a. use some of the Eclipse code short-cuts to build the classes quickly!
    b. create two instances of your shape
        i. one that works based solely of off invariant values
        ii. another that DOESN'T works based solely of off invariant values
            1. show that either the compiler or run-time doesn't like it
4. You will most likely be emailing your code, please have you and your teammates name in a comment at the head of each file

Differences between Abstract & Implements

- the differences will make your decision on which to use easier
- there are some nuances that can really… make a difference
  - and piss you off (as a programmer)

Differences Between Abstract and Implements		
Feature	**Abstract**	**Implements**
types of methods	Abstract-Non Abstract	Abstract
member variables	freedom of declaration	default to final
subclass uses to connect	"extends"	"implements"
members in general	freedom of specifier	public by default
overall length of code	more freedom, more code	very short
inheritance	subclass can only inherit 1 class (abstract class)	subclass can implement as many interfaces we want

Which one of the two options gives more freedom?
Try to finish the chart below. Place an X under which option it pertains to.

Which option supports:	Abstract	Implements
All methods declared within an _____ class ***must be*** implemented by the class(es) that implements this.		
You can define non-static or non-final field(s) in _____ class, so that via a method you can access and modify the state of Object to which they belong.		
You can expect that the classes that extend an _____ class have many common methods or fields, or require access modifiers other than public (such as protected and private).		
You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.		
Can house and support common code within subclasses		

Answers$_b$:

Multiple Inheritances

- sounds great huh?
  - but can be complicated
- means the structure and data members are being inherited by ***multiple base*** classes
- with Abstract
  - the subclass can only inherit 1 class (and it would have to inherit the abstract class)
- with Interfaces
  - there is not a limit to the number interfaces a class can implement

```
public class A implements C,D
public class A extends B implements C,D{...}
```

Solutions

Abstract Employee

```
import java.util.Date;

public abstract class Employee
{
	private String name;
	private Date hireDate;

	public abstract double getMonthlyPay();
	// abstract since this method will be defined differently in the sub classes
	// both sub-class definitions need to return a double and be of the same name

	public Employee(String name, Date hireDate)
	{
		this.name = name;
		this.hireDate = hireDate;
	}

	public boolean samePay(Employee other)
	{
		return (this.getMonthlyPay() == other.getMonthlyPay());
	}

// setters and getters

	public boolean equals(Object obj)
	{
		return false;
//
	}
}
```

SalariedEmployee

```
public class SalariedEmployee extends Employee {
  private double salary; // annual

  public SalariedEmployee(String name, Date hireDate, double salary) {
    super(name, hireDate);
    this.salary = salary;
  }

// setters and getters
  public double getMonthlyPay() { return getSalary() / 12; }

  @Override
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    long temp;
    temp = Double.doubleToLongBits(salary); // please, whatevs
    result = prime * result + (int) (temp ^ (temp >>> 32));
    return result;
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (!super.equals(obj))
      return false;
    if (getClass() != obj.getClass())
      return false;
    SalariedEmployee other = (SalariedEmployee) obj;
    if (Double.doubleToLongBits(salary) !=
Double.doubleToLongBits(other.salary))
      return false;
    return true;
  }

  @Override
  public String toString() {
    return "SalariedEmployee [salary=" + salary + ", name=" + name + ",
hireDate=" + hireDate + "]";
  }

  public double getSalary() { return salary; }

  public void setSalary(double salary) { this.salary = salary; }

}
```

HourlyEmployee

```
public class HourlyEmployee extends Employee {
  private double wageRate;
  private double hours; // for the month

  public HourlyEmployee(String name, Date hireDate, double wageRate, double
hours) {
    super(name, hireDate);
    this.wageRate = wageRate;
    this.hours = hours;
  }

  // setters and getters

  @Override
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    long temp;
    temp = Double.doubleToLongBits(hours);
    result = prime * result + (int) (temp ^ (temp >>> 32));
    temp = Double.doubleToLongBits(wageRate);
    result = prime * result + (int) (temp ^ (temp >>> 32));
    return result;
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj) {
      return true;
    if (!super.equals(obj))
      return false;
    if (getClass() != obj.getClass())
      return false;
    HourlyEmployee other = (HourlyEmployee) obj;
    if (Double.doubleToLongBits(hours) != Double.doubleToLongBits(other.hours))
      return false;
    if (Double.doubleToLongBits(wageRate) !=
Double.doubleToLongBits(other.wageRate))
      return false;
    return true;
  }

  @Override
  public String toString() {
    return "HourlyEmployee [wageRate=" + wageRate + ", hours=" + hours + ",
name=" + name + ", hireDate=" + hireDate
        + "]";
  }

  public double getWageRate() { return wageRate; }

  public void setWageRate(double wageRate) { this.wageRate = wageRate;  }
```

```
  public double getMonthlyPay() { return getHours() * getWageRate() * 4; }

  public double getHours() { return hours; }

  public void setHours(double hours) { this.hours = hours; }

}
```

Which option supports:	Abstract	Implements
All methods declared within an _____ class ***must be*** implemented by the class(es) that implements this.		X
You can define non-static or non-final field(s) in abstract class, so that via a method you can access and modify the state of Object to which they belong.	X	
You can expect that the classes that extend an _____ class have many common methods or fields, or require access modifiers other than public (such as protected and private).	X	
You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.		X
Can house and support common code within subclasses	X	

Sources

CMSC 202 Notes – Polymorphism II

Abstraction in Classes
http://www.youtube.com/watch?v=jq3si0S8UIU
http://www.youtube.com/watch?v=AU07jJc_qMQ

Abstraction in Methods
http://www.youtube.com/watch?v=-PqkqRDFHk8

Interfaces
http://www.youtube.com/watch?v=AU07jJc_qMQ at 7:30 ish
https://www.youtube.com/watch?v=fX1xNMBTMfg

Basic Terminology and Examples
http://ecomputernotes.com/java/what-is-java/what-is-java-polymorphism
http://www.tutorialspoint.com/java/java_polymorphism.htm

Differences between Abstract and Implements
https://www.geeksforgeeks.org/difference-between-abstract-class-and-interface-in-java/