Session Overview

- Object Oriented Technology
- UML Class Diagrams
- Class Diagram: Animals
- Inheritance
- Implicit and Explicit Casting of References
- Aggregation and Composition
- Geometry Example
- Polymorphism
- Method Overriding
- Method Overloading

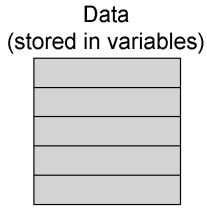
Introduction to Objects

- Objects are data and code linked together.
- The data is encapsulated within the object and methods (somewhat like functions) are used to access and modify the data.
- Different kinds of objects are represented with *classes*.
- Java is very object-oriented (although not perfect—there are primitive data types).

Data Encapsulation

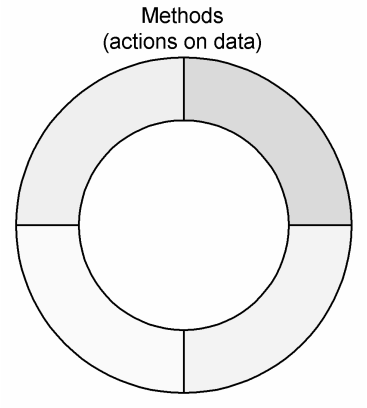
Data

- each object has its own state
- the state of an object is the current values of all of its variables
- the ***data*** portion of objects is stored in variables within an object
- typically, these variables are kept ***private***—they are not *directly* accessible from code outside the object.



Methods

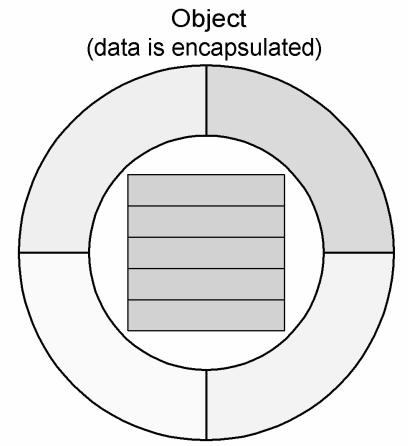
- ***methods*** are used to operate on the variables of an object
- a method is ***invoked*** by the caller
- methods are similar to functions, procedures, or subroutines in other languages—the main difference is that the method is invoked on a specific object



- some methods allow for ***parameters*** to be passed in; some don't take any parameters.
- some methods provide a ***return value*** at the end of execution; some don't return anything.
- methods that retrieve data from an object are called ***accessors***
  - sometimes, they are more loosely called ***getters***
- methods that alter data within an object are called ***mutators***
  - sometimes, they are more loosely called ***setters***

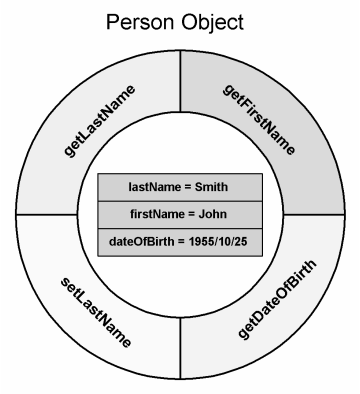
Objects

- ***objects*** combine data and methods into a unit
- each object has its own data (set of variables)
- the data is stored in the internal variables
- the data is said to be ***encapsulated*** within the object



Example: Person

- in this example, a Person object stores a particular person's
  - last name
  - first name
  - date of birth
- this data is encapsulated in a private zone that only the object's own methods can access or modify



- there are accessor methods to retrieve each of these values:
  `getLastName`
  `getFirstName`
  `getDateOfBirth`
- there is just one mutator method used to change the person's last name:
  `setLastName`

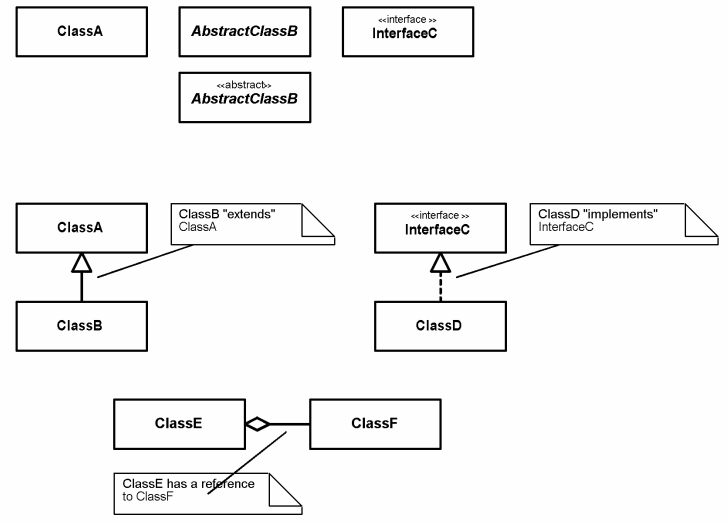
Messages

- *messages* are sent to an object from another object

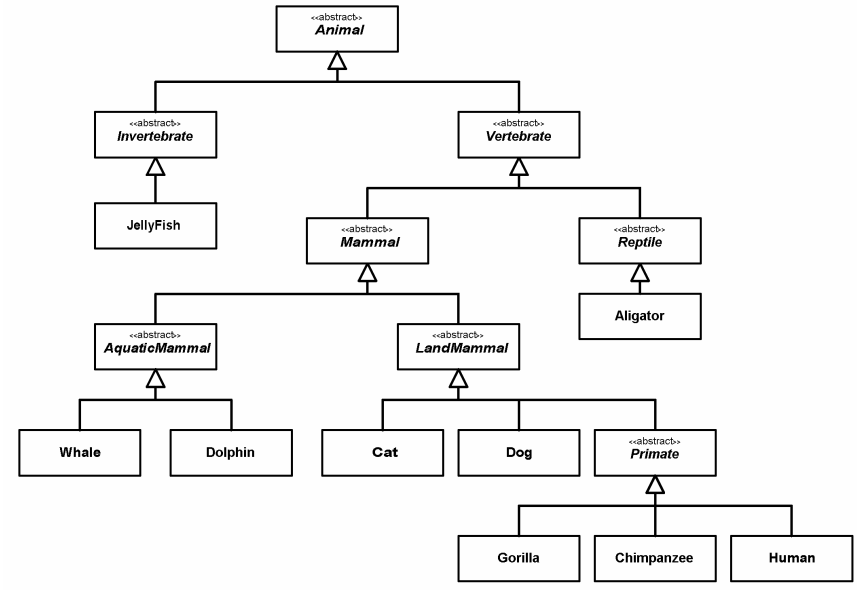
Classes vs. Objects

- objects are instances of a class
- there is one Person class
- there can be many Person objects

Unified Modeling Language (UML) Class Diagram Symbols



Example Class Diagram: Animals



- At the top of this class hierarchy, is the class named **`Animal`**
- As we move from top to bottom, we get more specific
- In this example, all of the intermediate layers happen to be abstract, but this is not a requirement of Java.

- To create an ***instance*** of a class (or to instantiate it, or to make an object), we do something like this:

```
Human john = new Human("John Smith");
```

  - the variable `john` is a ***reference*** to an instance of the class `Human`.
  - we construct an object—or instance—by accessing a ***constructor*** of the class `Human` with the `new` operator.

- To create another instance (object) of the class `Human`, do this:

```
Human jane = new Human("Jane Doe");
```

  - now there are ***two objects*** of the type `Human`, one has the name of "`Jane Doe`" the other "`John Smith`".
  - but there is only ***one class*** named `Human`.

Inheritance

- Classes that ***extend*** other classes ***inherit*** attributes and are considered ***subclasses*** of the class they extend.
  - for example, `Vertebrate` inherits characteristics from `Animal`.
  - and `Vertebrate` is a ***subclass*** of `Animal`.

- a ***superclass*** of a class is a class above it—or an ***ancestor***
  - for example, `Animal` is a superclass of `Vertebrate`.

- all of the ancestors of a class are considered the superclasses of that class, and often a class has more than one superclass.
  - for example, the superclasses of `Reptile` are `Vertebrate` and `Animal`.
  - the ***immediate*** superclass of `Reptile` is simply `Vertebrate`.

- all of the descendants of a class are considered the subclasses of that class, and there can be any number of subclasses—including zero.
  - the subclasses of `LandMammal` are: `Cat`, `Dog`, `Primate`, `Gorilla`, `Chimpanzee`, and `Human`.

- An instance of a class can be referred to by a reference variable of the type of *any of its superclasses*

```
Mammal m = new Human("Jane Doe");
Animal a = m;
```

- But, a object can't be referred to by one of its subclass types without explicit, checked casting:

```
Mammal m = new Human("Jane Doe");
Primate p = (Primate) m; // this works
Dog d = (Dog) m; // this casting to a subclass fails
```

  - the casting failure results in a **`ClassCastException`** being thrown.
  - objects know their real types and check that all assignments are valid.

- The **`instanceof`** operator is used to check if an object can be safely cast to a subclass:

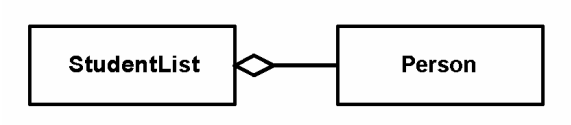
```
Mammal m = new Human("Jane Doe");

if ( m instanceof Primate ) {
    Primate p = (Primate) m; // this works
}
```

  - the result of an **`instanceof`** operation is **`true`** or **`false`**.

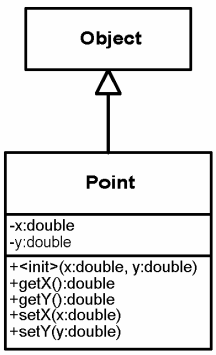
Aggregation and Composition

- objects can contain references to other objects
- if the objects referred to live and die with the referrer, then the relationship is called ***composition***.
- if the objects referred to can exist outside the context of the referrer, then the relationship is call ***aggregation***.
- In the following class diagram, a StudentList object contains at least one reference to a Person object:

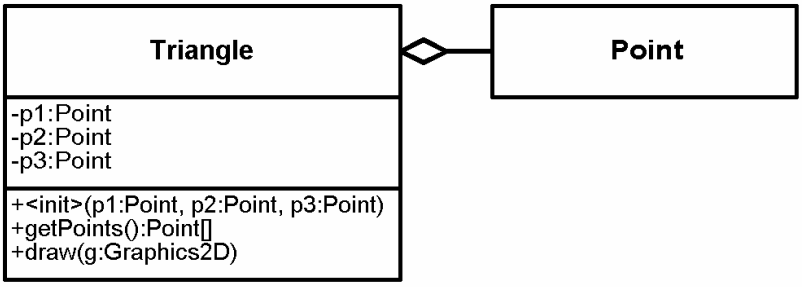


Geometry Example

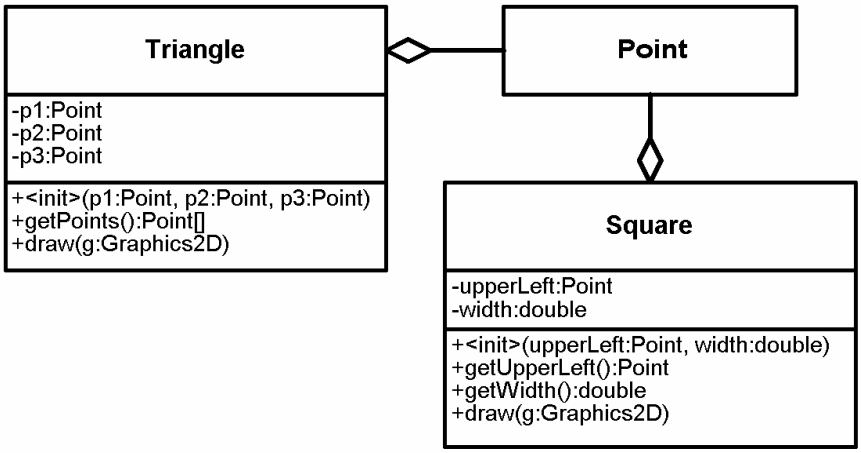
- a **Point** extends Object (all Java classes must have Object as one of their superclasses).
- a Point object holds a x and y value in a floating point primitive type called `double`.
- Point has a constructor that take the initial value for x and y (shown with <init> in the diagram)
- there are accessors and mutators for both x and y.



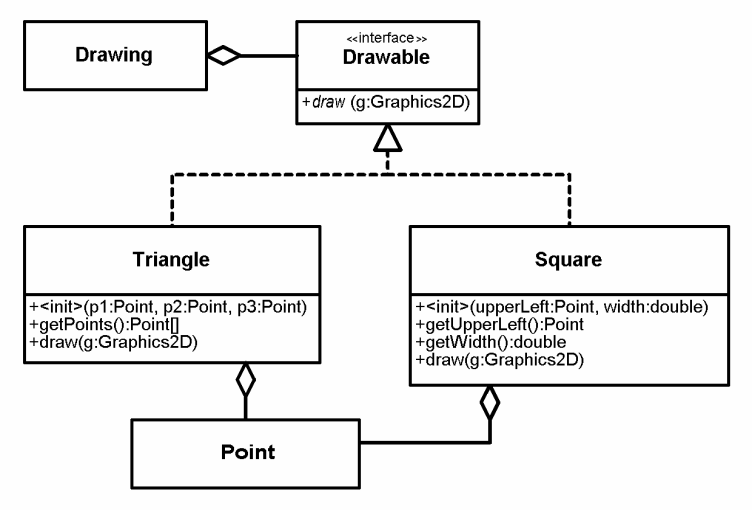
- a **Triangle** extends Object too, but this is not shown in this UML Class Diagram.
- a Triangle object holds three references to Point objects
- a Triangle will draw itself onto the passed Graphis2D context



- a **Square** also contains a reference to a Point object.



- What if we wanted to create a class called Drawing that could contain any number of Triangle, Square, or future shapes and be able to easily draw all of them?



Polymorphism

- when the same message can be sent to different types of objects
- in the geometry example, the draw(g:Graphis2D) method is polymorphic—its behavior changes depending on what the exact shape is.

Method Overriding

- a subclass can override a method from one of its superclasses by including a method whose signature matches the superclass's exactly.
- a method signature consists of the method name along with the type and order of the passes parameters.
- when an instance of the subclass has the overridden method called, the new behavior occurs.

Method Overloading

- when a class contains two or methods that have the same name, but different parameters
- example:

```
int valA = Math.min(4, 5);
double valB = Math.min(2.6, 11.4);
```