UML Tutorial 2 : Class Diagrams

Class Diagrams describe your programs classes and how they relate. An object created from a class is referred to as an instance of the class.

In Visual Paradigm we'll design our Class Diagram :

1. Diagram Navigator -> Select Class Diagram
2. Drag a Class onto the interface
3. Click on Class and Open Specification
4. In General -> Name: Animal, Visibility: public
5. Attributes (States / Fields)
    1. name : Visibility: protected, Initial Value: "", Type: String, Unique, Getter, Setter
    2. height : protected, double, 0.0, Unique
    3. weight : protected, double, 0.0, Unique
    4. favoriteFood : protected, String, "", Unique
    5. speed : protected, double, 0.0, Unique
6. Operations (Behaviors / Methods)
    1. getName
    2. setName
    3. (Rest of the setters & getters)
    4. Animal : Return: void, Visibility: public, Scope: instance, Unique
    5. move : void, public, instance, Unique
    6. eat : void, public, instance, Unique

When defining classes we want to focus on defining Abstract types. We must only use Fields and Methods that every Animal would possess. This is called Abstraction.

Encapsulation is the act of protecting our data like we did here with name. In the diagram public items begin with a +, protected a # and private a -. Then we list name, type and initial value. For methods we define name, parameters and return type.

Public means accessible to any code that can create an object. Protected means class methods and subclasses. Private fields can only be accessed in the class and not subclasses.

Show Multiplicity (Right Click Class > Presentation Options > Configure Class Presentation Options > Show Multiplicity. (Show Property Modifiers to Display Read only) Multiplicity refers to attributes that represent a group of objects. Set a range of values, or * for an unknown number. We also define whether values are all unique, ordered, or readOnly (Can't Change Value). Static fields are underlined.

Add the Following Attributes

1. favoriteFood : Food [1..10]
2. friends : Friend[*] {unique}
3. owners : Owner[*] {ordered}
4. favNum : int {readOnly}
5. numOfDogs : int (To define as static set Scope to classifier)

Inheritance

Drag the Resource Catalog Button from Animal and select the inherited class option. Name the new class Dog. We define inheritance with non-filled arrow. Then we add new attributes and methods for the Dog class.

We'll add the protected attribute breed of type Breed with getters and setters to Dog. Add a constructor that receives the Dog breed. Define a public method named bark that returns a String.

Associations

Create a Composition connection between Dog and the new class Breed. We create a compositional association because Dog owns the attribute breed.

Draw an Aggregation link between Breed and Dog. This signifies that Dog is the Aggregate of many other objects that describe it like Breed does.

Constraints

Draw a Constraint line from Dog. Constraints define rules for your classes. Here we'll say that when a Dog is created its breed can't be empty. self.breed -> notEmpty()

We can also define Pre & Post Conditions. Give Dog an attribute amtOwed which is protected and a double. It holds the value owed when purchasing a Dog. We'll define a Pre Condition that amtOwed must be greater than zero and a Post Condition that it must be zero after setOwner() is called.

setOwner()
pre: amtOwed > 0
post: amtOwed = 0

Abstract Classes

Create a new class association with Dog. Name it ShowAnimal. Right click ShowAnimal > Model Element Properties > Abstract. Now define a method in ShowAnimal named doFancyStuff. Mark it as Abstract as well. Abstract methods must be defined in the class that inherits it which is noted with italic text.

Interfaces

Interfaces contain only abstract methods. Attributes are either static or constants. Click the small black triangle next to the Class tool and select Interface. Name the Interface SuperAnimal and give it an operation fly that returns a String. Drag from Dog to SuperAnimal to define the connection and put fly in Dog.