Handout 6

Part 1: Defining Services

A Service is a specific behavior that each Object in a Class is responsible for exhibiting. (a.k.a Methods)

Services serve two main purposes:
1. Define required behavior
2. Define the necessary communication between objects, animate the model

Standard Services (required of all Class&&Objects):
   (Algorithmically-Simple Services)
   - Create an object (construction)
     ⇒ when passed data necessary to create an object, check that the data satisfies listed in the model constraints;
       if constraints are met – create a new instance of an object
   - Release an Object (destruction)
     ⇒ disconnect and delete an Object
   - Access an Object
     ⇒ Get (read) and Set values of Object’s Attributes.
       Otherwise known as accessor and mutator methods.
   - Connect an Object
     ⇒ Connect an Object with another

Some also require Algorithmically-Complex Services

- Calculate
  ⇒ Calculate a result derived from Object Attribute values

- Monitor
  ⇒ Monitor an external system or device.
    Monitoring usually implies services Initialize and Terminate (the device or process)
Services and Inheritance

A Child Class (Subclass) inherits all Services of its Parent (Superclass). It may introduce its own implementation of the Service (Overriding). Otherwise, the Parent’s implementation of the Service is going to be used each time the Service is requested from a Child Class.

A Child Class may have other Services than those defined in the Parent.

An Abstract Superclass may not contain an actual implementation of the Service. (Why?) In this case each Subclass must have its own implementation.

Part 2: Encapsulation and Polymorphism

Encapsulation revisited

Attributes and Services together encapsulate Object state and its evolution over time:

Attributes describe the state
Services change the state

Benefits of such encapsulation

Since Attributes can only be accessed via Services such representation creates a strong information hiding mechanism – cannot change the object’s state arbitrarily, can only “ask” the Object itself to change it. This increases robustness of software.

It also enforces model cohesion by keeping related content together in one place (Object Class).

Polymorphism revisited

Different Object Classes may share a common Service which is implemented differently in each class. This is typical for Objects that share a common Superclass.

For instance all DrawingObjects can perform Service Draw, yet Lines, Circles and Rectangles are drawn differently, thus must use a Draw method specific for each.

Each Subclass inherits all Services (as well as Attributes) of a Superclass. Yet, each Subclass may provide its own implementation (or, if you will, version) of the Service. In such case a Subclass is said to Override a method of the Superclass.