**Object Oriented Computing**

**Basics**

* An *Object* is an entity containing data that responds to messages
* A *Class* describes the characteristics of similar objects and has two main parts
  + Variables: define an objects data (internal state)
    - Instance variable – data belonging to a single object (things different from individual to individual)
    - Class/Static Variable – data that is shared among every object of the class
  + Methods: define an objects behavior and how it communicates
    - Instance Method – responds to message sent to one object
    - Class/Static method – responds to message sent to class

**Object Oriented Features**

* Primary features:
  + Encapsulation
  + Inheritance
  + Polymorphism
* *Encapsulation-* hiding and restricting access to the internal state and implementation details of an object from the user
  + also known as information hiding

**Enforcing Encapsulation**

* OO languages allow restrictions on the level of access to data and methods:
  + Public – access allowed to all users of the class
  + Private – access allowed only within the class itself
  + Default – no modifier. Accessed from the same class, accessed from the same package
  + Protected – access allowed only within the class itself or any classes which inherit from the class
* Public methods, their return types, and parameters make up an object’s interface

**Object Interfaces**

* Interface – description of how communication occurs between an Object and its user
* Three types of interface methods:
  + Constructor – creates an instance of a class (object)
  + Accessor – Requests information form an object
  + Mutator – Alters the state of an object

**Object Oriented Design**

* The principle of OO programming is to build software based on objects rather than functionality
* Common mistakes
  + Calling something a class when in reality it is

**Introduction to UML**

* The Unified Modeling Language

**Diagram Types**

* Diagrams describe specific aspects of the system
  + Use-case
  + Class
    - Most common way to describe design of OO systems
  + Object
  + State
  + Sequence

**UML Class representation**

* Classes are represented with boxes separated into three parts:
  + Class name
  + Variables
  + Methods
* *Identifiers are preceded by + for public, # for protected, and – for private*
* Type follows identifier name, separated by :
* Static variables/methods are underlined

**Inheritance**

* *Inheritance* – allowing one class to reuse the attributes and behavior of another class
* The class being inherited from is the *Superclass*
  + Also called a parent or base class
* The *Subclass* inherits all of the variables and methods form the Superclass
* However, it can only access the *public* and *protected* variables and methods

**Forms of Inheritance**

* Specification (Interfaces)
  + Superclass only defines the behavior
  + Subclass is responsible to implement the behavior
  + In java, a class can implement multiple different interfaces
* Specialization
  + The subclass is a specialized form of superclass
  + The subclass satisfies the specs of the superclass in all relevant aspects
  + The subclass inherits both the specs of a behavior and all/some of the behaviors provided by the superclass
  + This is the other form of inheritance used in Java
  + Extension:
    - Subclass adds new functionality
    - Cannot change inherited behavior
  + Limitation
    - The subclass restricts the use of some of the behavior inherited from the superclass
      * Removes behaviors inherited by the superclass
  + Combination (multiple inheritance)
    - The subclass inherits features from the multiple superclasses
    - Java does NOT support this form of inheritance

Inheritance Hierarchy

* The relationship between superclasses and subclasses is known as an inheritance hierarchy
* In Java, all classes are in an inheritance hierarchy because they all have the class “Object” as a superclass
* Allows more versatility

Constructor Chaining

* The parent’s constructor is called before the child constructor executes when the child constructor is called

Interface Classes

* Some Abstract Classes implement no behavior or data but just contain a list of methods for subclasses to implement
* These are called interface classes

Composition in UML

* Often an object will contain other objects
* This is shown with a line and note about how many of each object is contained
  + Diamonds colored in= composition (name belongs to a single instance of an individual)
  + Diamonds not colored in = aggregation (address *may* change, affecting all involved)

Address

Person

Name

Abstract Classes

* Some classes can never be instantiated because they don’t fully implement all of their behavior
* These are called Abstract Classes
* They exist for the sole purpose of defining attributes and behavior for classes to inherit
* Abstract classes in UML are written in italics, along with all of its abstract methods
* Abstract classes are similar to reminders that the compiler will tell you, making sure that you create concrete methods based off of the abstract classes abstract methods
* Example:

Public *abstract* class Shape

{

Private int x;

Private int y;

Public *abstract* int getArea();

}

Polymorphism

* Polymorphism – ability of a method or operator to have different behaviors based on context
* Overloaded methods
  + Same name but different parameters
* Overridden methods
  + Same name and parameter but different behavior in subclass
* Subclass assignment
  + Can assign object of subclass type to a variable of superclass
* Dynamic Method Binding (AKA late binding)
  + Determines which potentially overridden method to use for an object when subclass assignment may have been used

Comparison of Objects

* For Objects to be ordered, they must be comparable (i.e numbers, strings, etc.)
  + Some objects have no logical ordering
    - Cars can’t be sorted in ascending or descending order
* Classes that do have a logical ordering must implement the compareTo interface

Cloning

* Much like == only tests references, using “=” to copy an Object only copies the references
  + Example:
    - Student bob = new Student();
    - Student bob2 = bob;
      * Now, both bob and bob2 point to the same place in memory
* To make an actual copy of an Object you must use the clone method
  + Example:
    - Student bob = new Student();
    - Student bob2 = bob.clone();
      * Now, bob and bob2 are separate objects with the same data

Persistent Data

* Data is considered persistent if it exists from one execution of a program to another
  + This data must be saved to a file between executions
* Java makes this quite simple with the Serializable interface
  + Classes that implement the Serializable interface can be written with an ObjectOutputStream and read with an ObjectInputStream
  + Classes that require special handling must override the methods writeObject and readObject methods.