How to Draw a Class Diagram:

A **class diagram**, also called a **static diagram**, shows the relationships between classes and interfaces. These relationships are static – that is, they exist before any objects are instantiated, even before a program is executed. (An object diagram, on the other hand, shows the contents of memory at a given point in the execution of a program. An object diagram is dynamic – changing as the program executes.

The idea of a class diagram is part of the **Unified Modeling Language (UML)**, a powerful set of tools for designing andspecifying object-oriented programs. This note offers a suggested basic set of rules for drawing class diagrams in this course. More subtle variations are possible, and you are welcome to use any valid system as long as it expresses, at a minimum, the relationships defined here, and you are consistent. Here are the basic elements of a class diagram:

**Class Blocks:** A block is used to represent each class and interface. The block is made up of 3 subsections:

1. The first subsection shows the name of the class or interface, and possibly a qualifier, like <<abstract>> or <<interface>>.
2. The second subsection lists instance variables (attributes).
3. The third section lists methods (behaviors or operations).

It is quite common for the information provided about attributes and operations to be incomplete. A basic rule to follow in this course is to show whatever information is important to support the way the diagram will be used. Here are some possible variations:

- Visibility of a member may be indicated by preceding its name with a symbol: + for public, # for protected, or – for private.
- Certain (maybe even all) instance variables may be omitted. For example, perhaps only public instance variables are shown. One note: inherited instance variables are never shown – they appear elsewhere in the diagram (in the superclass). When instance variables are omitted, it is common to show an ellipsis (…) to indicate this.
- Certain (maybe even all) methods may be omitted. Again, it is common to show an ellipsis (…) to indicate this.
- Inherited methods are not shown unless they are overridden.
- When constructors are shown, they should be located before other methods.
- The data types of variables may be shown, or not, depending on the application, but this should be done consistently.
- The return types of methods may be shown, or not, depending on the application; this should also be done consistently.
- If they are important to the purpose of the diagram, class constants may be shown before instance variables.
"IS-A" Relationships: The fundamental purpose of a class diagram is to show the relationships between classes. The strongest relationships are "IS-A" relationships, and are represented by lines with large, closed arrowheads. Use a solid line to represent inheritance ('extends'), and a dashed line to represent 'implements'.

A extends B: \[ \text{A} \rightarrow \text{B} \]
C implements D: \[ \text{C} \rightarrow \text{D} \]

"HAS-A" and "USES-A" Relationships: When a class makes use of another class, a dependency at some level exists. If class E has an instance variable of type F, we say E "HAS-A" F. If class G has a method with a formal parameter of type H, we say G "USES-A" H (perhaps a weaker relationship than "has-a"). Both of these types of relationships are considered to be weaker than "IS-A" relationships. We use an arrow with a smaller, open arrowhead to represent these relationships. You may choose to use a solid line for "HAS-A" and a dashed line for "USES-A", or you may use a solid line for both.

E "HAS-A" F: \[ \text{E} \rightarrow \text{F} \]
G "USES-A" H: \[ \text{G} \rightarrow \text{H} \]

Here is an example:

Code:
```java
public interface Treasure extends SimThing {
}
public abstract class Seeker implements SimThing {
}
public class Vehicle extends Seeker {
    private Treasure gems; ...
}
```

Relationships:
- Seeker "IS-A" SimThing
- Seeker implements SimThing
- Treasure "IS-A" SimThing
- Treasure extends SimThing
- Vehicle "IS-A" Seeker
- Vehicle extends Seeker
- Vehicle "HAS-A" Treasure