Managing Objects:

- Often we need to maintain objects in memory in our application so we can easily store/retrieve then using a single piece of information.
- When the application terminates, we have two options:
    1. We do a fresh start the next time.
    2. We store the results of the computation so they can be used next time when the application runs.
- Say we have defined a class Student where every instance does have a property ID which is unique.
  We need the capability to read previously entered Student objects and store them into a Map.
  Once we have completed processing of the existing Student objects and may add a few more, we want to persist Student objects from the Map into a file.
  Suppose that the StudentManager class is responsible for:
    1. Reading the data from a csv file.
    2. Constructing Student objects based on data from the csv file and populating a Map with those Student objects.
    3. Before the application terminates, write the data to a file.

  If a CSV file is used, then the next time the application is launched, we would need to parse the file and reconstruct the student objects by calling on the Student constructor and passing in arguments.
  Rather than writing the values of an object's instance variables to file, a representation of the object itself can be written to file.
  An object is **serializable** if it can be represented as a sequence of bytes.
  The serialized object can be written to file.
  The object can later be deserialized. That is, the object can be reconstructed using the data read from the file.
- Java provides the interface Serializable to serialize objects.
- In order for a class to be serializable, it and its ancestor(s) must implement the Serializable interface and every instance variable in the class must also be Serializable.
- All of Java's primitive types are serializable.
- **Logging** is the process of recording events that occur during execution of a program in a central location.
- Messages may be written to a log file or to another location such as the standard error stream, System.err.
- java.util.Logger provides logging capabilities.
- When we generate a log message, we give it a level of severity:
    - SEVERE (highest)
    - WARNING
    - INFO
    - CONFIG
    - FINE
    - FINER
    - FINEST (lowest)
- By default, only messages with level INFO or higher are shown. But we can control that threshold using setLevel.
- A **handler** receives messages from the logger and either writes them to file or the console, or passes them on to be handled elsewhere.
- In Java, Handler classes include:
    - ConsoleHandler (sends log messages to stderr)
    - FileHandler (sends log messages to a file).

Inversion of Control (IOC):

- An object-oriented abstract design, also called a **framework**, consists of an abstract class for each major component.
- Frameworks can be built on top of other frameworks by sharing abstract classes.
- The framework often plays the role of the main program in coordinating and sequencing application activity.
- This **inversion of control** gives frameworks the power to serve as extensible skeletons.
- The methods supplied by the user tailor the generic algorithms defined in the framework for a particular application.
- IOC means that rather than having the application call the methods in a framework, the framework calls implementations provided by the application.
- Inversion of control is sometimes facetiously referred to as the "Hollywood Principle: Don't call us, we'll call you".
- **Dependency Injection:** The control on how dependencies are acquired by client classes resides in the underlying injectors/DI framework(s).
- **Observer pattern:** The control is transferred from the observers to the subject when it comes to reacting to changes.
- **Template method pattern:** The template method design pattern defines the skeleton of an algorithm in the superclass but lets subclasses override specific steps of the algorithm without changing its structure. Hence, control resides in the base class that defines the template method, instead of in the subclasses, implementing the algorithm's steps.