# Overview:

* The Flyweight pattern uses sharing to support a large number of fine-grained objects efficiently.
* The pattern is primarily used to reduce the number of objects created.
  + Less number of objects reduces the memory usage.
  + Memory usage is also minimized by sharing data as much as possible.
    - Crucial for low memory devices, such as mobile devices or embedded systems.
  + Performance is also increased.
* Tries to reuse already existing similar kind objects by storing them.
  + One instance of a class can be used to provide many “virtual instances”.
  + Creates a new object when no matching object is found.
* Flyweight objects are shared and are immutable.
  + Cannot be modified once they have been constructed.
* Flyweight objects are used in multiple contexts simultaneously and act as an independent object in each context.
  + Indistinguishable from an instance of the object that is not shared.

# Examples:

* Suppose two people were each searching for an apartment so that they could stay nearby their office:
  + Neither of them was satisfied with the available options.
  + One day, they found a place with all kind of facilities that they both desired.
    - There were two constraints:
      * There is only one apartment.
      * The rent is high.
  + Therefore, they decided to stay together and share the rent.
* The graphical representation of characters in word processors is a common example of this pattern.
* A computer game where we have a large number of participants
  + Looks are the same but differ from each other in their performances (or color, dresses, weapons…).
* All the wrapper classes valueOf () method uses cached objects.
  + Java String class String Pool implementation.

# Intrinsic vs extrinsic state:

* Two common terms are used when learning about the Flyweight pattern:
  + Intrinsic state/properties: can be stored in the Flyweight object and is shareable.
  + Extrinsic state/properties: depends on the Flyweight’s context and is not shareable.
    - Client objects define state and pass the extrinsic state to the Flyweight.
* Let’s look at an example that demonstrates the differences between these two terms:
* A text editor application where we enter characters:
  + An object of Character class is created.
  + The attributes of the Character class are name, font, and size.
  + We do not need to create an object every time a client enters a character since letter ‘B’ is no different from another ‘B’.
* If a client again types a ‘B’ we simply return the object which we have already created before:
  + All of these are intrinsic states (name, font, size).
  + They can be shared among the different objects, as they are similar to each other.
* If we add more attributes to the Character class:
  + Row and column
    - Specify the position of a character in the document.
  + These attributes will not be similar even for the same characters.
    - No two characters will have the same position in a document.
  + These states are termed as extrinsic states and cannot be shared amongst objects.

# Advantages and drawbacks:

* Reduces the number of object instances at runtime.
  + Saves memory.
* Centralizes state for many “virtual” objects into a single location.
* Can control many instances for a class in the same way.
* One drawback is that single, logical instances of the class will not be able to behave independently from the other instances.

# When to use Flyweight:

* When an application uses a large number of objects.
* When storage costs are high because of the sheer quantity of objects.
* When relatively few shared objects may replace many groups of objects, (once extrinsic state is removed).
* When the application does not depend on object identity.
  + Since Flyweight, objects may be shared, identity tests will return true for conceptually distinct objects.