Singleton Pattern

- Ensure a class only has one instance, and provide a global point of access to it.
- Singleton pattern limits the number of instances to often one, but generally a limited number.
- Serialisation may make implementing singleton hard.
- The singleton pattern pretty much needs just a private constructor and a public static getter to a private static instance field.
- `enums` (in Java at least) removes the reflection based and serialisation based issues of singleton creation.

When to use Singleton Pattern

- There must be exactly one instance (or a limited number of instances) of a class, and it must be accessible to clients from a well-known access point.
- The sole instance should be extensible by subclassing and clients should be able to use an extended instance without modifying their code. You should be able to use either an object of the base class or the derived class, but there should be only one instance among them. `static` methods are not extensible, so cannot be used.
    - `enums` (in Java at least) remove the reflection based and serialisation based issues of singleton creation.
    - If you want extensibility, you have to implement singleton properly.

Consequences of using Singleton Pattern

- Controlled access to sole instance.
- Reduces name space.
- Permits refinement of operations & representation.
- Permits a variable number of instances.

- More flexible than class operations (class operations ⇒ static methods).

Singleton vs. Other Patterns

- Several patterns are implemented using Singleton Pattern.
- For instance, Abstract Factory needs a Singleton Pattern for single instance of the factory.