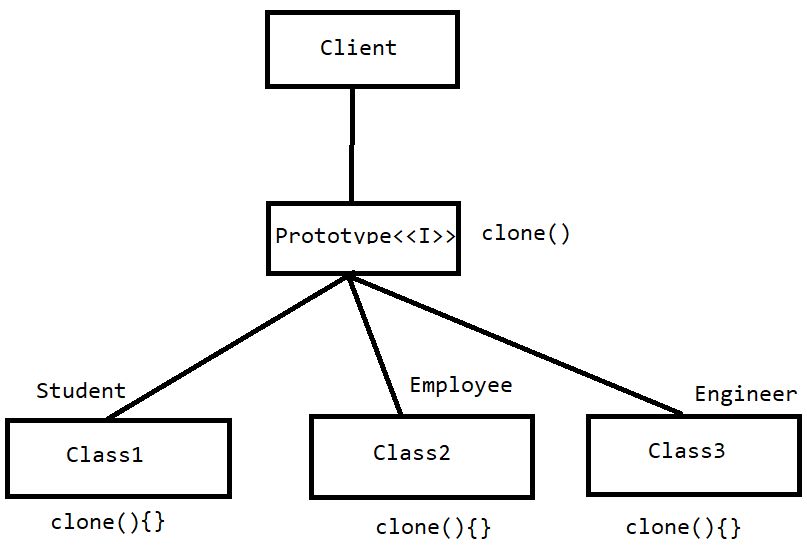
**Prototype Design Pattern:** The Prototype Design Pattern is a [creational pattern](https://www.geeksforgeeks.org/creational-design-pattern/) that enables the creation of new objects by copying an existing object. Prototype allows us to hide the complexity of making new instances from the client. The concept is to copy an existing object rather than create a new instance from scratch, something that may include costly operations. The existing object acts as a prototype and contains the state of the object.

* The newly copied object may change the same properties only if required. This approach saves costly resources and time, especially when object creation is a heavy process.
* The prototype pattern is a creational design pattern. Prototype patterns are required when object creation is a time-consuming, and costly operation, so we create objects with the existing object itself.
* One of the best available ways to create an object from existing objects is the clone() method. Clone is the simplest approach to implementing a prototype pattern. However, it is your call to decide how to copy existing objects based on your business model.
* For example: *Suppose a user creates a document with a specific layout, fonts, and styling, and wishes to create similar documents with slight modifications.*



**When to use the Prototype Design Pattern**

* **Creating Objects is Costly:**
  + Use the Prototype pattern when creating objects is more expensive or complex than copying existing ones.
  + If object creation involves significant resources, such as database or network calls, and you have a similar object available, cloning can be more efficient.
* **Variations of Objects:**
  + Use the Prototype pattern when your system needs to support a variety of objects with slight variations.
  + Instead of creating multiple classes for each variation, you can create prototypes and clone them with modifications.
* **Dynamic Configuration:**
  + Use the Prototype pattern when your system requires dynamic configuration and you want to create objects with configurations at runtime.
  + You can prototype a base configuration and clone it, adjusting the properties as needed.
* **Reducing Initialization Overhead:**
  + Use the Prototype pattern when you want to reduce the overhead of initializing an object.
  + Creating a clone can be faster than creating an object from scratch, especially when the initialization process is resource-intensive.

**When not to use the Prototype Design Pattern**

* **Unique Object Instances:**
  + Avoid using the Prototype pattern when your application predominantly deals with unique object instances, and the overhead of implementing the pattern outweighs its benefits.
* **Simple Object Creation:**
  + If object creation is simple and does not involve significant resource consumption, and there are no variations of objects, using the Prototype pattern might be unnecessary complexity.
* **Immutable Objects:**
  + If your objects are immutable (unchangeable) and do not need variations, the benefits of cloning may not be significant.
  + Immutable objects are often safely shared without the need for cloning.
* **Clear Object Creation Process:**
  + If your system has a clear and straightforward object creation process that is easy to understand and manage, introducing the Prototype pattern may add unnecessary complexity.
* **Limited Object Variations:**
  + If there are only a few variations of objects, and creating subclasses or instances with specific configurations is manageable, the Prototype pattern might be overkill.

