Prototype design pattern

Key terms

Prototype

> The prototype pattern is a creational design pattern that can be used to create objects that are similar to each other. The pattern is used to avoid the cost of creating new objects by cloning an existing object and avoiding dependencies on the class of the object that needs to be cloned.

Prototype

> Prototype allows us to hide the complexity of making new instances from the client. The concept is to copy an existing object rather than creating 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 same properties only if required. This approach saves costly resources and time, especially when object creation is a heavy process.

Let us say we have to created a new `User` API and we want to test it. To test it, we need to create a new user. We can create a new user by using the `new` keyword.

```
User user = new User("John", "Doe", "john@doe.in", "1234567890");
```

We might be calling a separate API to get these random values for the user. So each time we want to create a new user we have to call the API. Instead, we can create a new user by cloning an existing user and modifying the fields that are necessary. This way we can avoid calling the API each time we want to create a new user. To clone an existing user, we have to implement a common interface for all the user objects `clone()`

```
public abstract class User {
    public abstract User clone();
}
```

Then we can create an initial user object which is known as the prototype and then clone it using the `clone()` method.

```
User user = new User("John", "Doe", "john@doe.in", "1234567890");
User user2 = user.clone();
user2.setId(2);
```

Apart from reducing the cost of creating new objects, the prototype pattern also helps in reducing the complexity of creating new objects. The client code does not have to deal with the complexity of creating new objects. It can simply clone the existing object and modify it as per its needs. The client code does not have a dependency on the class of the object that it is cloning.

Prototype Registry

The prototype pattern can be extended to use a registry of pre-defined prototypes. The registry can be used to store a set of pre-defined prototypes. The client code can then request a clone of a prototype from the registry instead of creating a new object from scratch. The registry can be implemented as a key-value store where the key is the name of the prototype and the value is the prototype object.

For example, we might want to create different types of users. A user with a `Student` role, a user with a `Teacher` role, and a user with an `Admin` role. Each such different type of user might have some fields that are specific to the type so the fields to be copied might be different. We can create a registry of pre-defined prototypes for each of these roles.

```
interface UserRegistry {
    User getPrototype(UserRole role);
    void addPrototype(UserRole role, User user);
}
```

Now we can implement the `UserRegistry` interface and store the pre-defined prototypes in a map.

```
class UserRegistryImpl implements UserRegistry {
    private Map<UserRole, User> registry = new HashMap<>();

    @Override
    public User getPrototype(UserRole role) {
        return registry.get(role).clone();
    }

    @Override
    public void addPrototype(UserRole role, User user) {
        registry.put(role, user);
    }
}
```

The client code can request a prototype from the registry, clone it, and modify it as per its needs.

```
UserRegistry registry = new UserRegistryImpl();
registry.addPrototype(UserRole.STUDENT, new Student("John", "Doe",
"john@doe.in", "1234567890", UserRole.STUDENT, "CS"));

User user = registry.getPrototype(UserRole.STUDENT);
user.setId(1);
```

Recap

- The prototype pattern is a creational design pattern that can be used to create objects that are similar to each other.
- Recreating an object from scratch can be costly as we might have to call an API to get the values for the fields or to perform some other costly operations. The prototype pattern can be used to avoid this cost by cloning an existing object and modifying the fields that are necessary.
- Also, the client code does not have to deal with the complexity of creating new objects. It can simply clone the existing object and modify it as per its needs.
- To implement the prototype pattern, we follow these steps:
  1. `Clonable interface` - Create a common interface for all the objects that can be cloned.
  2. `Object class` - Create a concrete class that implements the common interface and overrides the `clone()` method.
  3. `Registry` - Create a registry of pre-defined prototypes with `register` and `get` methods.
  4. `Prototype` - Create a prototype object and store in the registry.
  5. `Clone` - Request a clone of the prototype from the registry and modify it as per its needs.

Design patterns in different languages

Prototype

Python

- Prototype - I
- Prototype - II
- Prototype - III
- Prototype - Code

Javascript

- Prototype - I
- Prototype - II
- Prototype - III
- Prototype - IV