Factory Method

Patrón de diseño creacional

- Resuelve problemas de inicialización y referenciación de objetos.
- Dentro de este patrón de diseño se encuentran: Singleton, Factory Method y Abstract Factory.

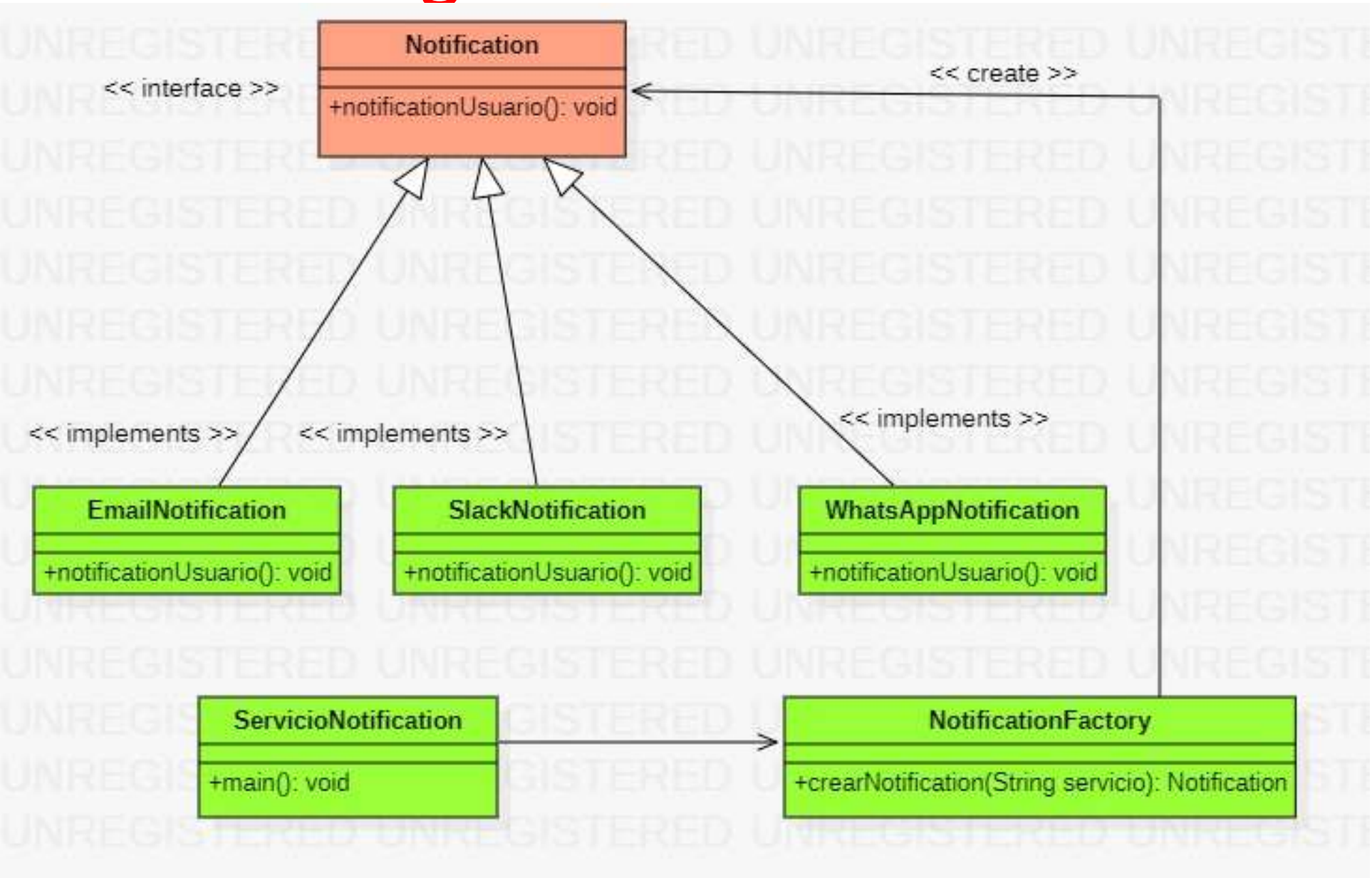
Principios de Factory Method

- Una clase delega la creación de objetos a sus subclases.
- El *creador abstracto* es una interface común para crear objetos donde se definen solo los métodos.
- Los *creadores concretos* (subclases) son clases que implementan la interface, decidiendo que objetos crear en función de los parámetros.
- Promueve la abstracción, la modularidad, la mantenibilidad y escalabilidad en el diseño del software.

Escenario de un problema real

- Suponer que las materias de las carreras de computación de la UNRC se quieren comunicar con sus alumnos a través de servicios de notificaciones como email, slack y whatsapp . . .

Diagrama de clases



Creador abstracto

```java
public interface Notification {
    public void notificationUsuario();
}
```

Creador concreto - Fabrica

```java
public class NotificationFactory {
    /**
     * Retorna un constructor de clase.
     * @param servicio un mensaje o cadena ("EMAIL" o "SLACK" o "WHATSAPP")
     * @return un constructor de clase.
     */
    public Notification crearNotificaction(String servicio) {
        switch (servicio) {
        case "EMAIL":
            return new EmailNotification(); // referencia a EmailNotification()
        case "SLACK":
            return new SlackNotification(); // referencia a SlackNotification()
        case "WHATSAPP":
            return new WhatsAppNotification(); // referencia a WhatsAppNotification()
        default:
            throw new IllegalArgumentException ("No se conoce el servicio " + servicio);
        }
    }
}
```

Productos (Implementaciones de la interfaz)

```java
public class EmailNotification implements Notification {
    @Override
    public void notificationUsuario() {
        System.out.println("La notificacion es por EMAIL.");
    }
}
```

```java
public class SlackNotification implements Notification {
    @Override
    public void notificationUsuario() {
        System.out.println("La notificacion es por SLACK.");
    }
}
```

```java
public class WhatsAppNotification implements Notification {
    @Override
    public void notificationUsuario() {
        System.out.println("La notificacion es por WHATS APP.");
    }
}
```

Cliente (Método que utiliza la fabrica)

```java
public class ServicioNotification {
    Run | Debug
    public static void main(String[] args) {
        NotificationFactory nf = new NotificationFactory();
        Notification n = nf.crearNotificaction("WHATSAPP"); // new WhatsAppNotification();
        n.notificationUsuario(); // invoca al metodo de WhatsAppNotification()
    }
}
```