NOTEPAD UYGULAMASI

Notepad uygulamamız bir menu bar içerisinde 3 farklı JMenu’den oluşmaktadır. Bunlar sırasıyla : File(New, Open, Save, Save As, Exit) Edit(Undo, Redo, My Undo) ve Format(Font, Font Size, Bold, Italic, Bold And Italic) menüleridir.



Kullanıcı Kılavuzu

File menüsüne tıklayınca çıkan alt menülerden:

New -> Yeni bir dosya oluşturulmasını sağlar

Open -> Bilgisayardan var olan bir dosyayı açar

Save - > Önceden açılmış bir dosyanın üzerindeysek dosyaya kaydetme işlemi yapar. Önceden açılmış bir dosya yoksa Save As olarak çalışır.

Save As -> Dosyayı farklı kaydetmek için kullanılır.

Exit -> Notepad uygulamasından çıkış yapar.



Edit menüsünde:

Undo – Yapılan işlemi geri alır

Redo – Yapılan işlemi geri getirir

My Undo – Command Design pattern kullanılarak bizim oluşturduğumuz bir undo işlevselliğidir



Format Menusunde:

Font: “Arial” ya da “Times New Roman” fontlarından birisini seçebilirsiniz

Font Size : Farklı font büyüklükleri seçebilirsiniz.

Bold: Yazılan metni kalın harflerle yazar.

Italic : Yazılan metni italik harflerle yazar.

Bold And Italic : Yazılan metni hem kalın hem de italik harflerle yazar.



Uygulamamızı tasarlarken 6 farklı design pattern kullandık. Bunlar :

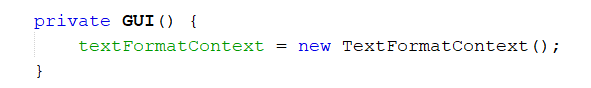
1. Singleton Design Pattern
2. Iterator Design Pattern
3. Template Method Design Pattern
4. Command Design Pattern
5. Factory Method Design Pattern
6. Strategy Design Pattern

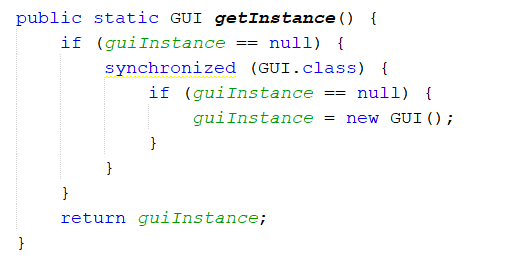
1) Singleton Design Pattern

Singleton Design Pattern , **Creational Design Pattern** grubunda yer alır. Singleton desgin pattern çalışma zamanında yalnızca 1 obje yaratılmasını garanti eden tasarım desenidir.

Uygulamamızda GUI class’ından oluşturulan nesnenin sadece bir tane olması gerektiğinden Singleton design pattern kullanılmıştır:







Birden fazla threadin olduğu ortamlar için her defasında sadece bir threadin işlem yapabilmesi adına synchronized() kullanıldı. Bu durumda gui intance’ımız çoktan yaratılmışsa bile her thread beklemek zorunda kalmasın diye “the double-checked locking idiom” kullanıldı. Gui instance’ımız partially constructed halindeyken başka bir process ona erişemesin diye volatile keywordü kullanıldı.

Nesne oluştururken getInstance() metodunun kullanımı Main class içinde yapıldı:

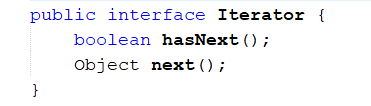


2) Iterator Design Pattern

Tekrarlayıcı tasarım kalıbı; birleşik bir nesnenin bileşenlerine, nesnenin esas ifadesinin gösterilimini açığa çıkarmadan sırayla erişebilmeyi sağlar. Tekrarlayıcı tasarım kalıbı, bir listenin yapısının ve çalışma tarzının uygulamanın diğer kısımları ile olan bağlantılarını en aza indirmek için; listede yer alan nesnelerin, sırasıyla uygulamadan soyutlanması amacıyla kullanılır. Biz bu kapsamda menülerimizdeki item namelerimizi dolaşabilmek adına iterator design patternı kullandık.

Tekrarlayıcı kalıbın içinde yer alan yapılar aşağıdaki gibi sıralanır:

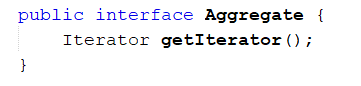
* **Tekrarlayıcı (Iterator):** Öğelere ulaşmak ve öğeleri çaprazlamak için bir arabirim tanımlar. (Uygulamamızdaki Iterator isimli interface)



* **Somut Tekrarlayıcı (Concrete Iterator):** Tekrarlayıcı arayüzünü sağlar. (Uygulamamızdaki MenuItemNameIterator class’ı)



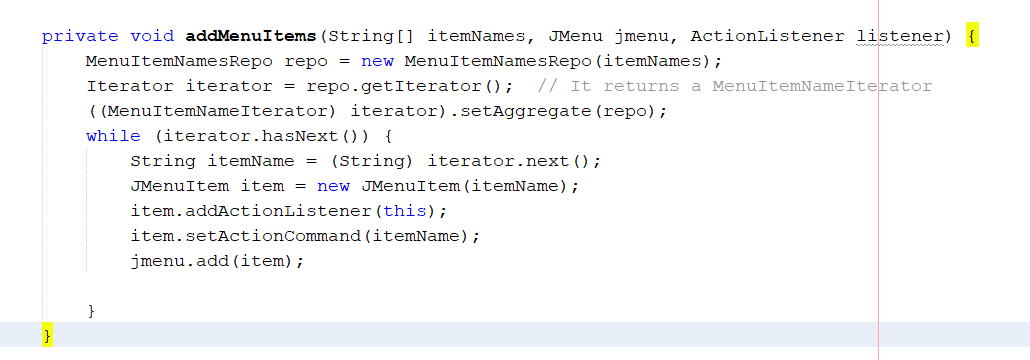
* **Birleşik (Aggregate):** Tekrarlayıcı nesnelerin yaratılması için bir arabirim oluşturur. (Uygulamamızdaki Aggregate interface’i)



* **Somut Birleşik (Concrete Aggregate):** Uygun somut tekrarlayıcı aşamasına dönmek için tekrarlayıcı arabiriminin oluşturulmasını sağlar. (Uygulamamızdaki MenuItemNamesRepo class’ı)

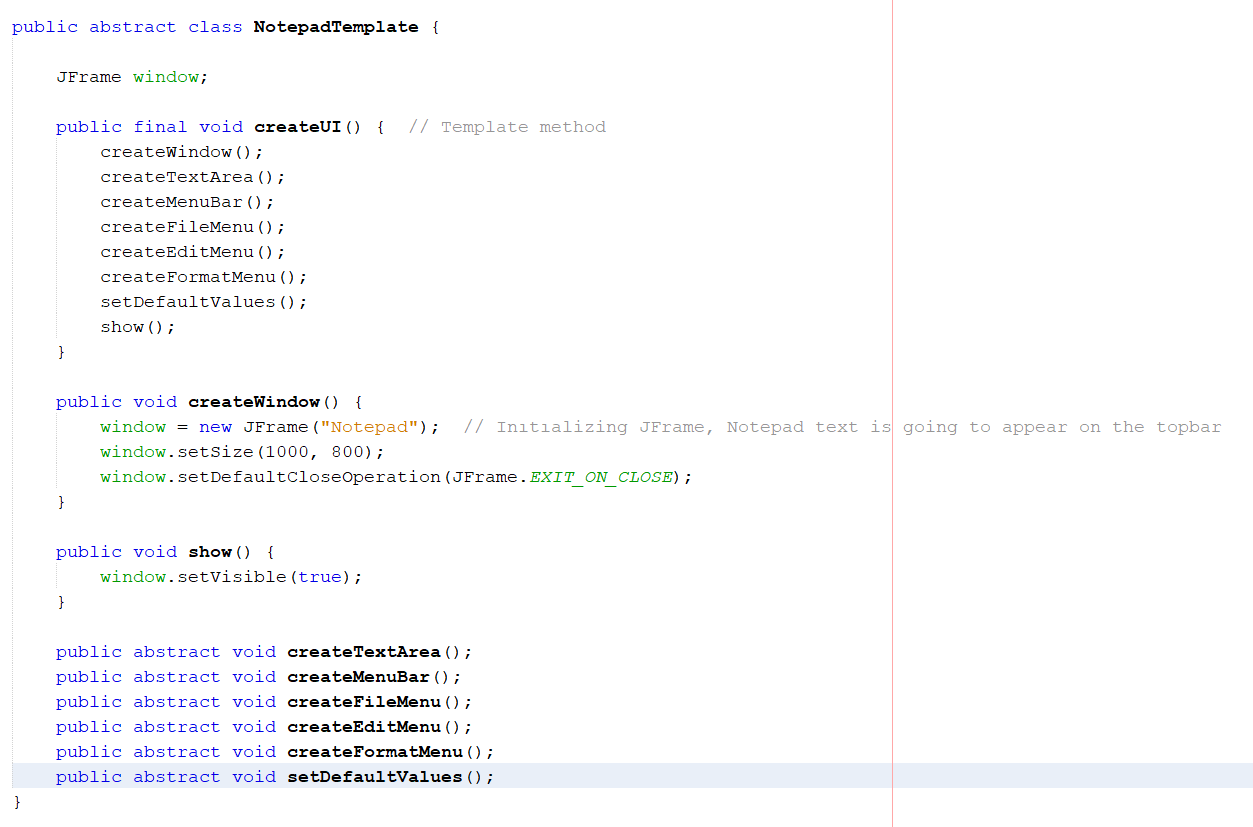


Iterator’ı kullandığımız yer ise GUI class’ımızdaki addMenuItems() metodu. Bu metod parametrede verilen item name’leri parametrede verilen JMenu’ye ekler. Ve verilen item name array’inin üzerinde gezinmek için iterator’ımızı kullanır:



3) Template Method Design Pattern

Şablon metot kalıbı davranışsal tasarım kalıpları grubunda yer alır. Şablon metot kalıbı bir işlem için gerekli adımları soyut olarak tanımlar ve bir şablon metot(template method) ile algoritmanın nasıl çalışacağını belirler. Alt sınıflar algoritma için gerekli bir ya da birden fazla metodu kendi bünyelerinde gerçekleyerek kullanılan algoritmanın kendi istekleri doğrultusunda çalışmasını sağlarlar. Template Method, akışın(ya da algoritmanın) ana iskeletini kendi elinde tutup detayların geliştirilmesini alt sınıflara bırakır. Uygulamamızda birden fazla Notepad UI’ı dizayn edilmek istenebileceğini düşünerek ve her dizaynda gerçekleştirilmesi gereken belirli adımlar olduğu için bu tasarım desenini kullandık. Bunun için NotepadTemplate isminde soyut template class’ımızı oluşturduk. Her yeni notepad UI tasarımında bir JFrame oluşturulması ve bu JFrame’in gösterilmesi isteneceği için createWindow() ve show() metodlarının gerçekleştirimini bu soyut sınıfta yaptık. Her notepad dizaynında farklılaşabilecek menü bar, file, edit, format menü ve text area tasarımlarını abstract method olarak tanımlayıp alt sınıfa bıraktık. Son olarak createUI() isimli template metodumuzda algoritmanın nasıl çalışacağını belirledik. Bu template’i extends eden GUI class’ımız ise soyut metodların gerçekleştirimini yaparak algoritmanın kendi isteği doğrultusunda çalışmasını sağladı.



Template methodunun kullanımı Main Class içinde yapıldı:,

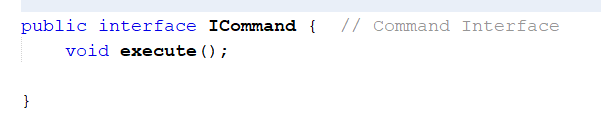


4) Command Design Pattern

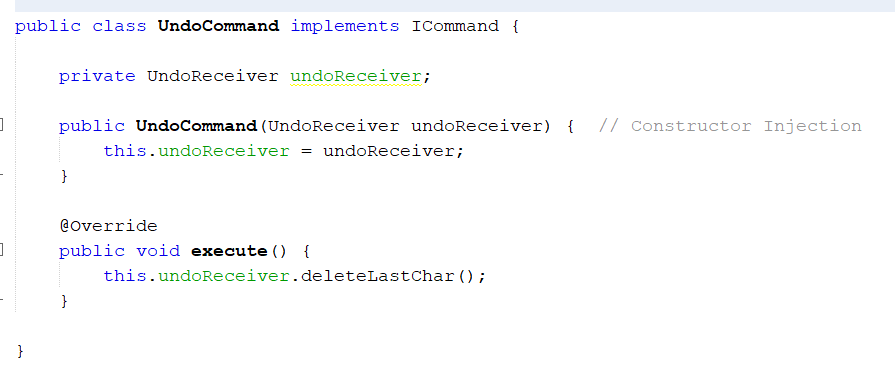
Command ( Komut ) tasarım deseni, bir isteği kendisi ile ilgili tüm bilgileri içeren bağımsız bir nesneye dönüştüren davranışsal bir tasarım desenidir. Komut tasarım kalıbı, kullanıcı isteklerini gerçekleştiren kod yapısının sarmalanarak nesneler halinde saklanmasına dayanır. Biz uygulamamızda bu deseni undo işlevini gerçekleştirmek için kullandık. Java’da UndoManager ile bu işlem hazır yapılabildiği için bunu da projemize ekledik fakat Edit menüsündeki My Undo item’ına tıklayınca geri alma işlemi bizim Command Design Pattern kullanarak yazdığımız kodlarla sağlanıyor.

Komut tasarım kalıbının kullanıldığı tüm durumlarda geçerli olan bazı ortak terimler mevcuttur.

* **Komut (Command) :** Gerçekleştirilecek işlem için bir ara yüz tanımlar. (Uygulamamızdaki ICommand interface’i)

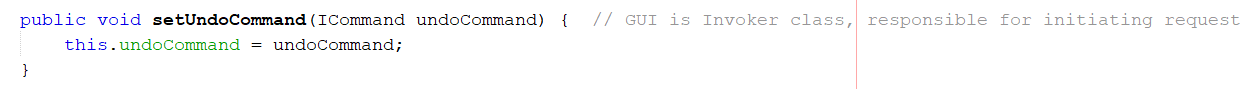


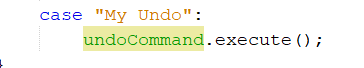
* **Somut Komut (Concrete Command):** Alıcı ve gerçekleştirilecek işlemler arasında bir bağ kurar, alıcıda karşılık düşen işlemleri çağırarak çalıştırma eylemini gerçekleştirir. (Uygulamamızdaki UndoCommand class’ı)



* **Çağırıcı (Invoker):** Metodun ne zaman çağrılacağını belirtir. (Uygulamamızdaki GUI class’ı)

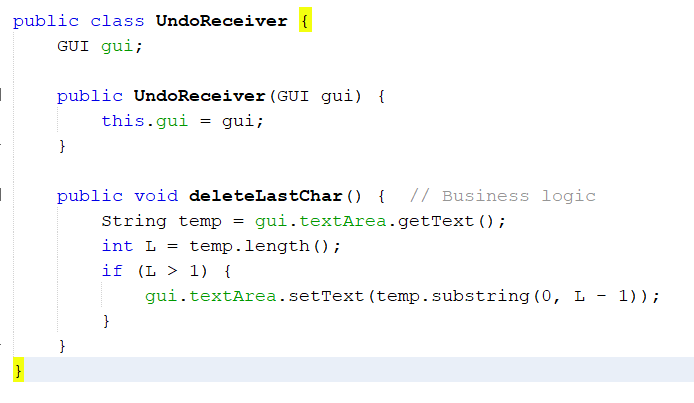






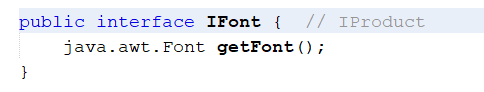
GUI class’ında actionPerformed() metodu yazılmıştır. My Undo item’ına tıklanınca metod çağrılır.

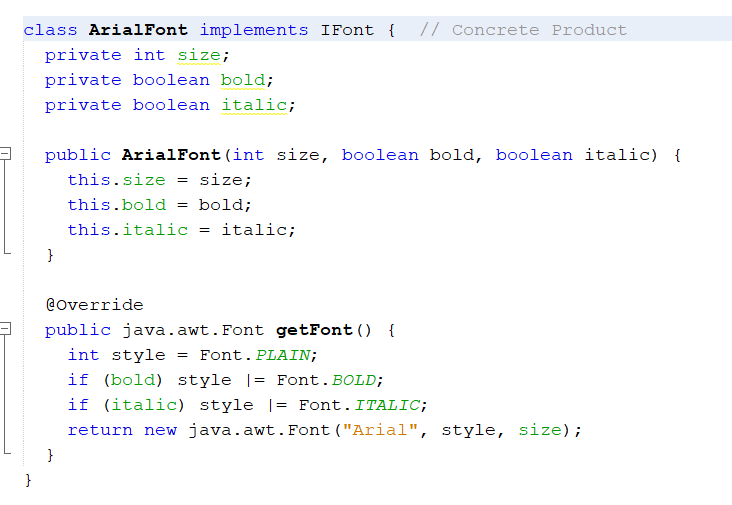
* **Alıcı (Receiver):**  Kullanıcı isteklerini gerçekleştirecek asıl metod kodlarını içerir. (Uygulamamızdaki UndoReceiver class’ı)



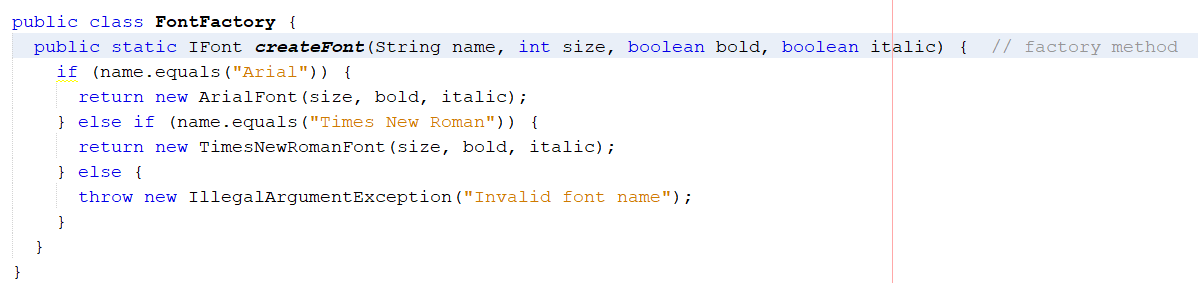
5) Factory Method Design Pattern

Kelime anlamı “Fabrika Metodu” olan Factory Method, üst sınıfta nesneler oluşturmak için bir arabirim sağlayan, ancak alt sınıfların oluşturulacak bu nesne türünü değiştirmesine izin veren bir yaratımsal desen (creational pattern) türüdür. Fabrika metodu ürün oluşturma kodunu, ürünü kullanan koddan ayırır. Böylece yeni ürün oluşturma kodları kodun geri kalanından bağımsız olarak yazılabilir. Biz uygulamamızda bu design pattern’ı farklı fon tipleri oluşturmak için kullandık. Bu design pattern’ı kullanmasaydık formatlama fonksiyonlarını tuttuğumuz FormatFunctions classımızda eklenmek istenen her yeni font için switch-case yapısı kullanmamız gerekirdi. Fakat bu nesne oluşturma işlemini bizim için factory method yapıyor.

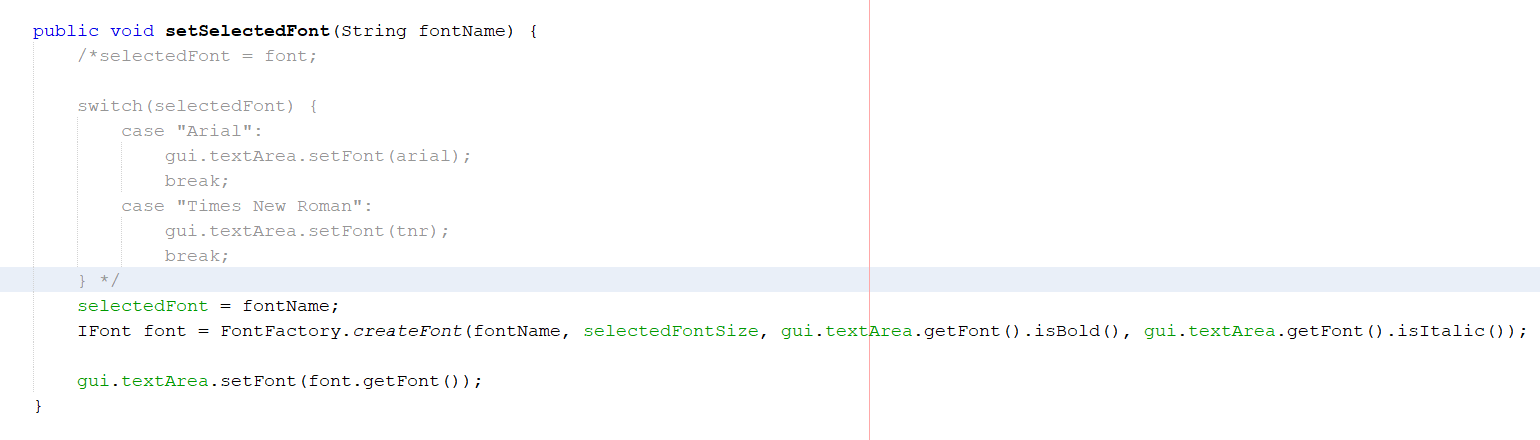








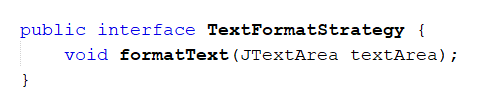
Kullanım:



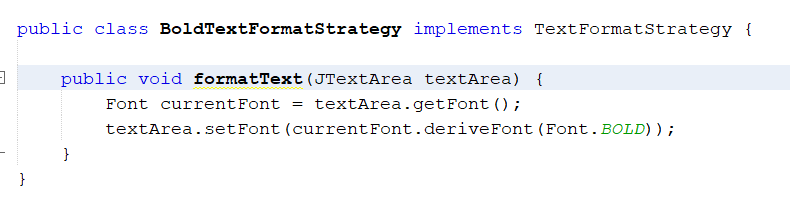
6) Strategy Design Pattern

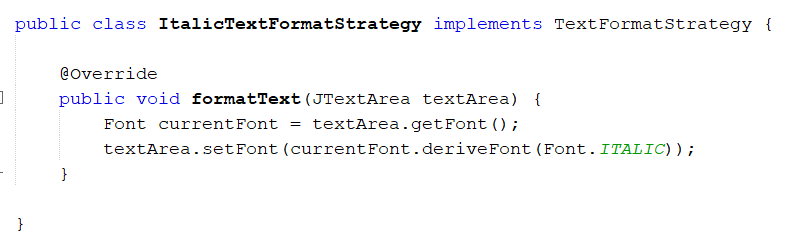
**Strategy** deseni bir algoritma ailesi oluşturup her birini farklı sınıfa yerleştirerek nesnelerini birbiri arasında değişebilir hale getirmeyi sağlayan bir tasarım desenidir. Bir fonksiyonun birden fazla yapılış şekli olduğu takdirde, bu fonksiyonelliği farklı versiyonlarıyla kullanmak istendiğinde kullanılabilecek bir design patternidir.

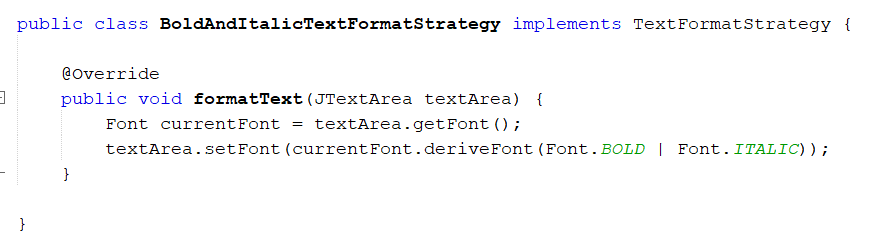
Aynı işi farklı şekillerde yapan birden fazla concrete strategy classımız olduğunda bunları bir strategy class üzerinden clienta sunmak, strategy classına da bu concrete tiplere ait ortak ata olan interfaceyi vermek , ilerleyen zamanlarda bu concrete tiplere bir yenisi daha eklendiğinde , işimizi kolaylaştıracak, bu durumda tek yapmamız gereken bu concrete tipi ortak interfaceden türetmek yeterli olacaktır. Uygulamamızda formatText() fonksiyonumuz seçilen format Bold, Italic, Bold and Italic için farklı gerçekleştirimlere sahip olduğu için Strategy Design Pattern’ı kullandık.



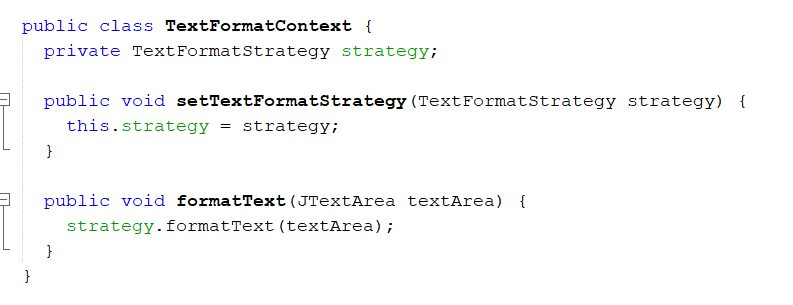
ConcreteStrategy class’larımız:



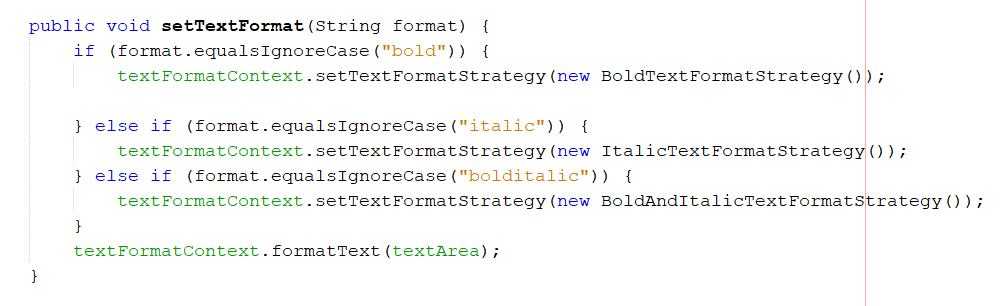




Context Class’ımız:

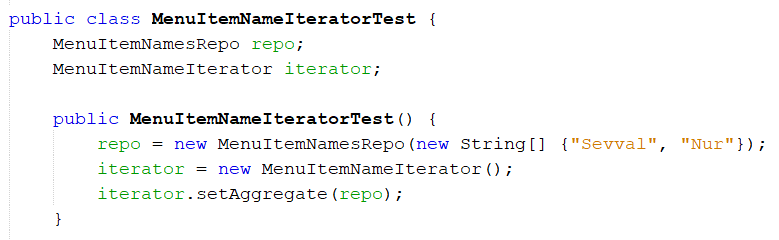


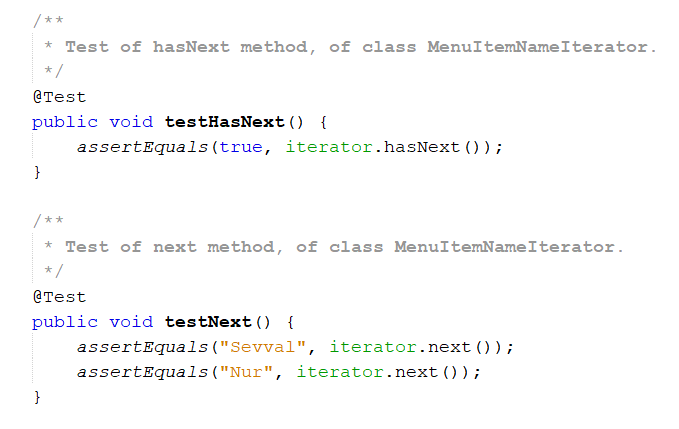
Kullanım:



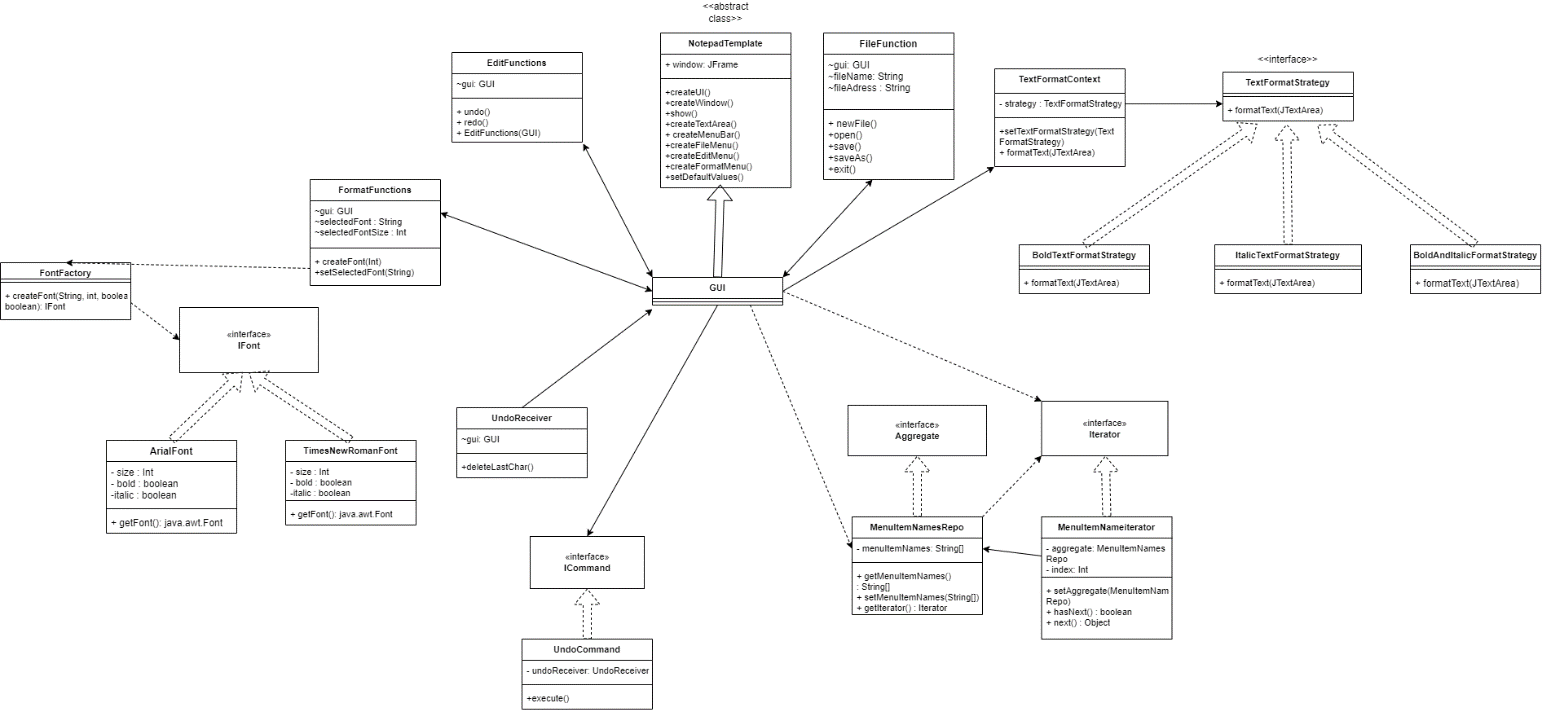
JUNIT TESTLERİ

Uygulamamızda MenuItemNameIterator class’ımızı test etmek için JUNIT testleri yazdık:





UML



Hazırlayanlar:

Şevval GÖNÜL – 05200000045

NOOR ABUBAKR - 05200000850