SPECULATIVE GENERALITY

Problema: Hay veces en que un codigo es creado “solo por si acaso” para tratar de dar apoyo a funcionalidades que se podrian implementar en un futuro, pero a la larga nunca es implementado y como consecuencia el codigo se vuelve mas dificil de entender o de modificar.
Tecnica de Refactorizacion: **Inline Class,** en este caso se elimina la interfaz ya que no aporta nada al programa y se remueven sus implementaciones**.**
Beneficios: Eliminar clases innecesarias libera memoria de la computadora y evita confunciones futuras

Codigo antes:

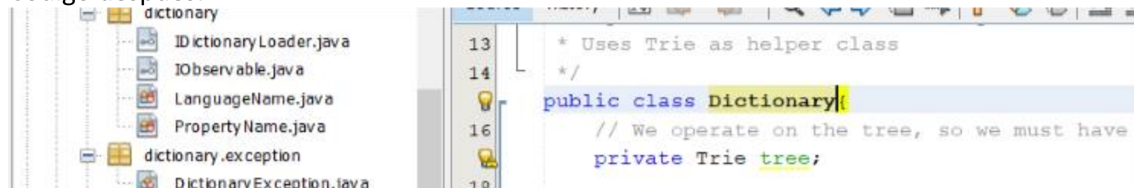
```
package edu.gu.hajo.dict;

/**
 * Interface to be used by other parts of application.
 * @author Kim Burgestrand
 */
public interface IDictionary
{
    // Methods missing
}
```

```
 * Uses Trie as helper class
 */
public class Dictionary implements IDictionary {
    // We operate on the tree, so we must have it! D:
    private Trie tree;

    /**
     * Load a given dictionary using the dictionaryloader provided.
     *
     * @param from
```

Codigo despues:



DATA CLASS

Problema: No se esta usando de manera efectiva el objeto, este podría hacer mucho mas.
Tecnica de Refactorizacion: **Encapsulate Field,** para que la clase tenga mas importancia en el programa, se hara privada la caracteristica que maneja y se crearan metodos para acceder a la misma.
Beneficios: Se pueden empezar a realizar operaciones complicadas que involucren el acceso a esta caracteristica del programa

Codigo antes:

```
    /**
     *
     * @param from, The from language
     * @param to, The to language
     * @return A map where key is a word from the from-language and
     *         the value is an array of translations of the key.
     * @throws DictionaryException
     */
    public Map<String, String[]> loadDictionary(LanguageName from, LanguageName to)
                                                throws DictionaryException;
}
```

Codigo despues:

```
    public Map<String, String[]> getLoadDictionary() {
        return loadDictionary;
    }

    public void setLoadDictionary(Map<String, String[]> loadDictionary) {
        this.loadDictionary = loadDictionary;
    }
```

PRIMMITIVE OBSESSION

Problema: Las clases podrían volverse gigantes e incambiables al momento en que se quieran añadir más funcionalidades o atributos
Técnica de Refactorización: **Replace data value with object**
Beneficio: Mejora la relación dentro de las clases. Los datos y los comportamientos relevantes están dentro de una sola clase.

Codigo antes:

```java
package edu.gu.hajo.dict;

/**
 *
 * THIS WILL BE USED IN LAB 2 as part of the MVC model
 *
 * These are the the names of the properties that's
 * fired by the dictionary implementation.
 * To be used by observers (GUI...) to determine what has changed
 * @author hajo
 *
 */
public enum PropertyName {
    FROM_LANG_CHANGED,
    TO_LANG_CHANGED

}
```

Codigo despues:

```java
public class PropertyName {

    private String Property;

    public PropertyName(String Property){
        this.Property = Property;
    }

     public String getProperty() {
         return Property;
     }

     public void setProperty(String Property) {
         this.Property = Property;
     }

}
```