Peer-graded Assignment: Capstone Assignment 2.3 – Identify and Fix Code Smells

1 Code Smell: Duplicate code

Class: ContactList and ItemList

Method:

```
public boolean saveContacts(Context context)
{
      try {
         FileOutputStream fos = context.openFileOutput(FILENAME, 0);
         OutputStreamWriter osw = new OutputStreamWriter(fos);
         Gson gson = new Gson();
         gson.toJson(contacts, osw);
         osw.flush();
         fos.close();
      } catch (FileNotFoundException e) {
         e.printStackTrace();
         return false;
      } catch (IOException e) {
         e.printStackTrace();
         return false;
      }
      return true;
   }
```

Why: Duplicated code is when you have blocks of code that are similar, but have slight differences. This block of code appears in multiple places and when it came time to make a change, you would have just needed to update it in one location. Having to update it in only one location will make it easier to implement the change and reduce the chance that you're missing a block of code that needs to be updated or changed.

Solution: We can create a aux class to implement this method to be called in **ContactList** or **ItemList** Class. In this method we can use a **generic type** and pass it as parameter and **reflection**.

```
public boolean saveContacts(Context context, String FILENAME, T xxxx)
{
    try {
       FileOutputStream fos = context.openFileOutput(FILENAME, 0);
       OutputStreamWriter osw = new OutputStreamWriter(fos);
       Gson gson = new Gson();
         // pass a T parameter
       gson.toJson(xxx.., osw);
       osw.flush();
       fos.close();
    } catch……
```

2 Code Smell: Switch Statements

Class: SectionsPagerAdapter

Method:

```
  @Override
  public CharSequence getPageTitle(int position) {
    switch (position) {
       case 0:
          return "All";
       case 1:
          return "Available";
       case 2:
          return "Borrowed";
    }
    return null;
  }
```

Why: There's a need to have big, long if else statements in your code. However, there are times that switch statements could be handled better. You want to be able to reduce these conditionals down to a design that uses polymorphism.

Solution:

```
public interface PageTitle{
    public CharSequence getPageTitle();
}
public class All implements PageTitle{
    public CharSequence getPageTitle(){
        return "All";
    }
}
public class Available implements PageTitle{
    public CharSequence getPageTitle(){
        return "Available";
    }
}
public class Borrowed implements PageTitle{
    public CharSequence getPageTitle(){
        return "Borrowed";
    }
}
```