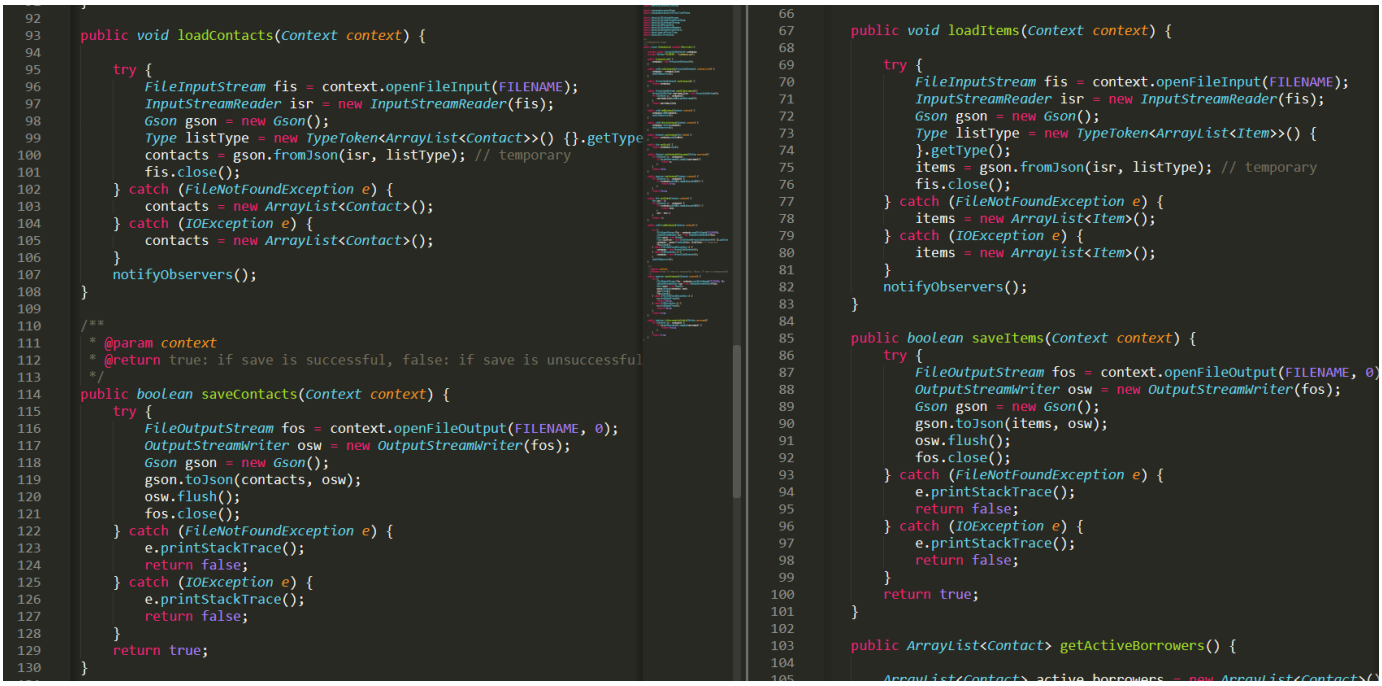
Capstone assignment 2.3

tags: `OOP` `Coursera`

Submission

Code Smell 1: Duplicated code



```
public void loadContacts(Context context) {

    try {
        FileInputStream fis = context.openFileInput(FILENAME);
        InputStreamReader isr = new InputStreamReader(fis);
        Gson gson = new Gson();
        Type listType = new TypeToken<ArrayList<Contact>>() {}.getType
        contacts = gson.fromJson(isr, listType); // temporary
        fis.close();
    } catch (FileNotFoundException e) {
        contacts = new ArrayList<Contact>();
    } catch (IOException e) {
        contacts = new ArrayList<Contact>();
    }
    notifyObservers();
}

/**
 * @param context
 * @return true: if save is successful, false: if save is unsuccessful
 */
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;
}
```

```
public void loadItems(Context context) {

    try {
        FileInputStream fis = context.openFileInput(FILENAME);
        InputStreamReader isr = new InputStreamReader(fis);
        Gson gson = new Gson();
        Type listType = new TypeToken<ArrayList<Item>>() {
        }.getType();
        items = gson.fromJson(isr, listType); // temporary
        fis.close();
    } catch (FileNotFoundException e) {
        items = new ArrayList<Item>();
    } catch (IOException e) {
        items = new ArrayList<Item>();
    }
    notifyObservers();
}

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

public ArrayList<Contact> getActiveBorrowers() {
```

- Location:

1. loadContacts(Context context) in class ContactList and loadItems(Context context) in class ItemList
2. saveContacts(Context context) in class ContactList and saveItems(Context context) in class ItemList

- Description:
  Despite the difference of the name of the function and the class they belong, these two pairs of functions have exactly the same codes. This means that when changes made to any of these functions, the changes will have to be done two times which is error prone.
- Solution: Create a class that handles storage by taking object type and name as its members. It handles reading and storing data.

Code Smell 2: Use too many Comments

Class EditContactActivity::saveContact(View view)

```
public void saveContact(View view) {

    String email_str = email.getText().toString();

    if (email_str.equals("")) {
        email.setError("Empty field!");
        return;
    }

    if (!email_str.contains("@")) {
        email.setError("Must be an email address!");
        return;
    }

    String username_str = username.getText().toString();

    // Check that username is unique AND username is changed (Note: if username was not changed
    // then this should be fine, because it was already unique.)
    if (!contact_list_controller.isUsernameAvailable(username_str) &&
            !(contact.getUsername().equals(username_str))){
        username.setError("Username already taken!");
        return;
    }

    // Reuse the contact id
    String id_str = contact_controller.getId();
    Contact updated_contact = new Contact(username_str, email_str, id_str);

    // Edit Contact: replace contact with updated contact
    boolean success = contact_list_controller.editContact(contact, updated_contact, context);
    if (!success) {
        return;
    }

    // End EditContactActivity
    finish();
}
```

Class EditItemActivity::toggleSwitch(View view)

```
/**
 * Checked == "Available"
 * Unchecked == "Borrowed"
 */
public void toggleSwitch(View view){
    if (status.isChecked()) {
        // Means was previously borrowed, switch was toggled to available
        borrower_spinner.setVisibility(View.GONE);
        borrower_tv.setVisibility(View.GONE);
        item_controller.setBorrower(null);
        item_controller.setStatus("Available");

    } else {
        // Means not borrowed
        if (contact_list.getSize()==0){
            // No contacts, need to add contacts to be able to add a borrower
            invisible.setEnabled(false);
            invisible.setVisibility(View.VISIBLE);
            invisible.requestFocus();
            invisible.setError("No contacts available! Must add borrower to contacts.");
            status.setChecked(true); // Set switch to available

        } else {
            // Means was previously available
            borrower_spinner.setVisibility(View.VISIBLE);
            borrower_tv.setVisibility(View.VISIBLE);
        }
    }
}
```

- Location:

1. saveContact(View view) in class EditContactActivity
2. toggleSwitch(View view) in class EditItemActivity

- Description:
  In these two functions, comments are spread before and over the whole functions. By percisly arrange the name of functions, classes and variables and by using enum and reference nicely, code can be read easily without redundant explaination made by comments. Excessive use of comments can cause extra work if the code changes. Also, outdated comments often occurs and misleads developers.
- Solution:
  Make variable name, method name self-explanatory. Use enumerators to define alias.