HIGH LEVEL DESIGN: https://github.com/CS3704-VT/Course/blob/main/Project/Design.md

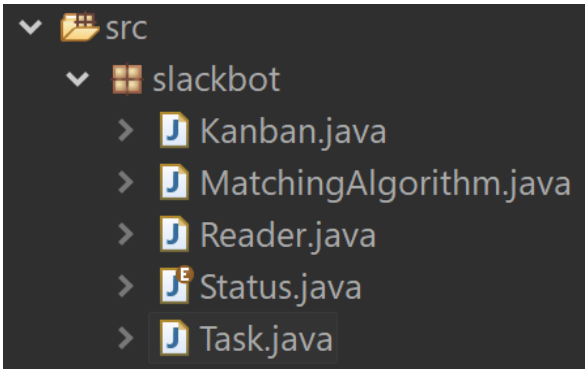
Event Based:

With our project being a ChatBot that tries to update Kanban boards, the event based design process would be best. It will show potential use cases that users of the system might try to perform and we'll be able to ensure that we anticipate how the system might be used. We could then also map how the system will react when the user gives it information and potential reactions. An example of the event based architecture working well with this project would be the bot being called based on the event of a user typing in the key words. This would be an example of detection and then a reaction from the event would occur. One of the pros of an event based architecture is that it supports the addition of new features which would help with an addition to the slack bot. One of the cons is that there would be no control on the order which is relevant to the slack bot since someone could try to move a non-existent task to a different category on the kanban board. This would need to be handled with error messages. Event based architecture would be the best fit since it is an event driven program.

LOW LEVEL DESIGN:

The three main design pattern families include creational, structural, and behavioral. Since creational deals with the process of object creation and the slack bot doesn't need to create more than a few classes, this would not be the most beneficial. Structural also includes the composition of classes, but our slack bot will only contain a few different objects like a message and return message, so this also doesn't fit for this project. The best design pattern would be **behavioral** since it characterizes the way classes and objects interact and distribute responsibility. This is the best pattern because our project needs to have a simple reader class that communicates with slack, the matching algorithm, and kanban board. This will also have a task object that will be passed between the classes that contains a string for the message, status, and string for participants assigned to the task.. This would be the most beneficial for the slack bot since it would be able to define the chain of responsibility, read the command, and change the state of the tasks.

Below is an example of code with multiple classes that interact with each other (Most of it is workable with a few parts that are not operational):



```java
package slackbot;

import java.util.ArrayList;

public class Kanban {

    private ArrayList<Task> todo;
    private ArrayList<Task> inpo;
    private ArrayList<Task> done;

    public Kanban() {
        todo = new ArrayList<Task>();
        done = new ArrayList<Task>();
        inpo = new ArrayList<Task>();

    }


    public void update(Task oldTask, Task newTask) {
        addTask(newTask);
        removeTask(oldTask);
    }


    public void addTask(Task task) {
        if (task.getStatus() == Status.DONE) {
            done.add(task);
        }
        else if (task.getStatus() == Status.TODO) {
            todo.add(task);
        }
        else {
            inpo.add(task);
        }
    }


    public void removeTask(Task task) {
        if (task.getStatus() == Status.DONE) {
            done.remove(task);
        }
        else if (task.getStatus() == Status.TODO) {
            todo.remove(task);
        }
        else {
            inpo.remove(task);
        }
    }


    public ArrayList<Task> getTodo() {
        return this.todo;
    }


    public ArrayList<Task> getDone() {
        return this.done;
    }


    public ArrayList<Task> getInpo() {
        return this.inpo;
    }
```

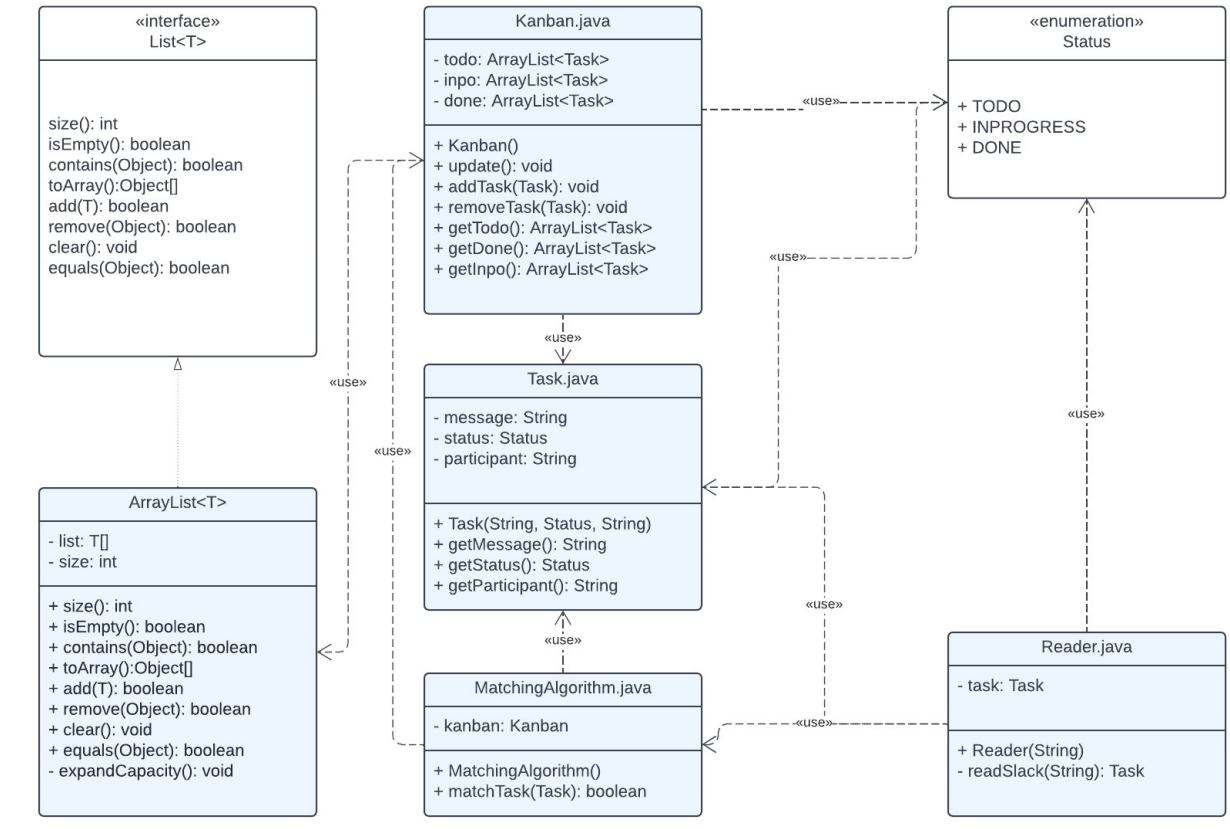
```java
package slackbot;

public enum Status {
    TODO, INPROGRESS, DONE
}

```

```java
package slackbot;

public class MatchingAlgorithm {

    private Kanban kanban;

    public MatchingAlgorithm() {
        kanban = new Kanban();
    }

    public boolean matchTask(Task task) {
        boolean found = false;

        // find the task in the kanban board and update the kanban
        // ...
        kanban.update(kanban.getDone().get(0), task);


        return found;
    }

}
```

```java
package slackbot;

public class Reader {

    Task task;

    public Reader(String slackChat) {
        MatchingAlgorithm ma = new MatchingAlgorithm();

        while (true) { // check the slackchat
            this.task = readSlack(slackChat);

            if (ma.matchTask(task)) {
                // print to slack success
            }
            else {
                // print to slack failure
            }
        }

    }


    private Task readSlack(String slackChat) {

        String message = "placeholder message"; // Connect to slack bot
        Status stat = Status.TODO; // Connect to the slack bot
        String participant = "name";

        Task task = new Task(message, stat, participant);

        return task;
    }
}
```

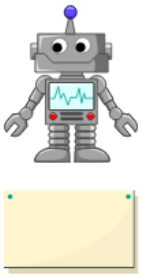
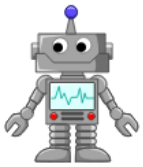
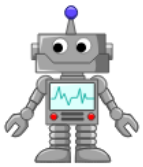
```java
package slackbot;

public class Task {
    private String message;
    public Status status;
    private String participant;

    public Task(String message, Status status, String participant) {
        this.message = message;
        this.status = status;
        this.participant = participant;
    }


    public String getMessage() {
        return this.message;
    }


    public Status getStatus() {
        return this.status;
    }


    public String getParticipant() {
        return this.participant;
    }
    // ...

}
```

Class Diagram:



DESIGN SKETCH:

***Storyboard** of the project leader creating and assigning a task to a team member using the bot.*

 Project leader enters command to create task (i.e. #create todo <task-name> <description>)	 Bot creates task and adds to Kanban board.	 Bot gives confirmation to project leader (i.e. “task added to board!”)	 Project Leader enters command to assign team member to task (i.e. #assign <user> <task>)
 Kanban board reflects assignment	 team member is notified of assignment (i.e. “New assignment: <task details>)	 Bot gives confirmation to Project Leader that the task has been successfully assigned	

As shown in the sketch, one design decision is to have only the project leader able to create and assign tasks, to prevent team members from reassigning work. Additionally it is the project leader’s job to keep track of the work that needs to be done, so they are the ones in control of creating tasks to disseminate. The other decision is the bot providing confirmation of commands and notifications such as successful task creation and assignment to allow for greater ease-of-use and communication.

11/10/23 PM3 Tyler Buxton, Tyler Kim, Patrick, Hunter, Krishna

AGILE PROCESS DELIVERABLE:

Hunter:
Yesterday: Received PM3 documentation; worked on P3
Today: Revised PM3, worked on storyboard
Stuck: Attempting to encapsulate all necessary results/interactions in storyboard

Krishna:
Yesterday: Worked on P3
Today: Looked over any errors in P3, double checked formatting
Stuck: Confirming the correctness of some sections

Patrick:
Yesterday: Wrote up the documentation for PM3 and passed it to the other team members.
Today: Worked on the high level design architecture for the Slack Chat Bot.
Stuck: Trying to determine what high level architecture is best for this Chat Bot.

Tyler Buxton
Yesterday: Worked on design implementations for the this project
Today: Worked on the high and low level documentation
Stuck: Stuck on figuring out when to have time to finish this project and others

Tyler Kim:
Yesterday: Came up with idea/outline for storyboard
Today: Worked on finishing a makeup tests and completing PM3
Stuck: Stuck working on the storyboard