# HOW DOES THE CODE FLOW ?

Okay, so we have got 4 x Packages under the main package Duke:

1. Command
2. Exception
3. Worker
4. Task

# Command package

## 1. Command (layout for any command you create → extend this)

This is an **abstract class** which you will **extend every time you create a new Command**

**EMPTY CONSTRUCTOR**

It has **two variables**:

```
protected Boolean exitRequest = false;
protected CommandType commandType;
```

It has **two methods**:

→ **Boolean getExitRequest()** **:** returns the exitRequet
→ **abstract void execute (TaskList tasklist) :** you choose what you do with this method

## 2. CommandType ( enum → list of User cmds possible )

**EMPTY CONSTRUCTOR**

So far we have the following Commands possible :

***TASK***,
***BYE***
***LIST***,
***BLANK***,
***FIND***,
***DELETE***,
***DONE***,
***QUEUE***,
***VIEWSCHEDULE***,
***REMINDER***

**It has one method:**

→ **static** String[] getNames() : returns a string array of the above enums to see if the user entered a cmd as listed

## 3. CommandBlank

**EMPTY CONSTRUCTOR**

**One method ( you override the execute method specified in Cmd):**

**void** execute **(TaskList tasklist)** → Prints a LINE ( _____)

## 4. CommandBye

**EMPTY CONSTRUCTOR**

**One method ( you override the execute method specified in Cmd):**

**void** execute **(TaskList tasklist)** → Sets the exitRequest to be true and prints the bye message

# 5. CommandDelete

**CommandDelete(String userInput) → HAS A CONSTRUCTOR**

**It has one variable :**

**protected** String **userInput**;

**One method ( you override the execute method specified in Cmd):**

→ **void** execute **(TaskList tasklist) :** uses the userInput and parses it to

- get the index of the task to delete
- deletes the task, after
- printing a message to the user

# 6. CommandError

**EMPTY CONSTRUCTOR**

**One method ( you override the execute method specified in Cmd):**

→ **void** execute **(TaskList tasklist) :** Prints the error message

# 7. CommandList

**EMPTY CONSTRUCTOR**

**One method ( you override the execute method specified in Cmd):**

→ **void** execute **(TaskList tasklist) :** Prints the no.of.tasks and all the tasks in the list

# 8. CommandFind

**CommandFind(String userInput) → HAS A CONSTRUCTOR**

**It has one variable:**

**protected** String **userInput**;

**It has two methods ( you override the execute method specified in Cmd):**

**void execute(TaskList taskList)** → Parses the userInput and then calls **findTasks**
**void findTasks(String name, TaskList taskList)** → loops through the task list to check if the taskname of every task contains the word which user entered

# 9. CommandMarkDone

**CommandMarkDone(String userInput) → HAS A CONSTRUCTOR**

**It has one variable:**

**protected** String **userInput**;

**It has three methods ( you override the execute method specified in Cmd):**

**void** execute(TaskList taskList) → Parses the userInput and marks the chosen task as done
→ Calls loadQueuedTasks

String genMarkDoneReply(**int** index, TaskList taskList) → Print statement of task which was marked as done

**void** loadQueuedTasks(TaskList taskList, Task mainTask) → checks if the task which was marked as done has tasks which are queued after it and if yes we append it to the list

# 10. CommandNewTask

**CommandNewTask(String userInput) → HAS A CONSTRUCTOR**

**It has two variable:**

```
protected String userInput;
protected TaskType taskType;
```

**It has two methods ( you override the execute method specified in Cmd):**

**void** execute(TaskList taskList) → Parses the userInput and creates a new task
→ Adds the task to the list

**void** hasForwardSlash(String input) → checks if the userInput has a '/'

# 11. CommandReminder

**HAS A CONSTRUCTOR**

```
public CommandReminder() {
        this.currentDate.setTime(0);
}
```

**It has one variable:**

```
protected Date currentDate = Calendar.getInstance().getTime();
```

**One method (you override the execute method specified in Cmd):**

**void execute(TaskList taskList) → Loops through the taskList and checks if currentDate matches the date associated with the task → if yes → print out**

# 12. CommandQueue - Ahmad Need to Vet

**CommandMarkDone(String userInput) → HAS A CONSTRUCTOR**

**It has one variable:**

**protected String userInput;**

## 13. CommandSchedule

**CommandSchedule(String userInput) → HAS A CONSTRUCTOR**

**It has one variable:**

**protected String userInput;**

**Two methods (you override the execute method specified in Cmd):**

**void execute(TaskList taskList) → Gets the date from the user input to print out the message followed by calling the method printSchedule to print out the schedule for the respective date**

**void printSchedule(String dateInput , TaskList taskList) → sets the time input to zero, loops through the tasklist to find all the tasks that has the input date specified and prints the respective tasks**

# Exception package

1. **DukeException**

   Throws the error message when it catches an exception.

# Worker package

## 1. Parser

Class to modify the user input such that it can be used by the other classes to carry out the designated commands.
This is the first class that the userInput goes through.

EMPTY CONSTRUCTOR

**Methods**

→ `private static String parseForEnum(String userInput, String[] enumTypes)`

- Loops through all the Enum commands added in by the developer and returns the enum that matches the userInput.
- It also checks the position of the enum in the userInput as, if the user was to create a DEADLINE named “event”, it might cause an issue if the program takes both as a command

→ `public static CommandType parseCommandType(String userInput)`

- Returns the value of type of command that the user has inputted.
- If there is no command it is interpreted as BLANK
- After iterating through all the possible enums, if it returns a blank, the command is taken to be TASK.

→ `public static Command parse(String userInput)`

- Has a switch case function which will compare the userInput with the present command types
- When matched, it will execute the COMMAND type from the command package and store the result in variable ‘c’ and returns it

→ `public static TaskType parseTaskType(String userInput)`

- After looping through all the commands, if the userInput doesn’t match any of the given enum, then it automatically given a dummy PARENT class
- This is in place to improve the user experience and not restrict too much on the userInputs

→ public static String removeStr(String commandStr, String userInput)

- Removes the first word of the userInput

→ static String encodeTask(Task task)

- Changes the data into a string, then encodes it so that is can be stored

→ public static String[] parseStoredTask(String encodedTask)

- Splits the encoded string at the given global marker variable (PARSE_MARKER_TASK)

→ public static String[] DecodeStoredTaskDetails(String taskString)

- Returns the array after decoding the already encoded and stored items

→ private static String encodeMainTask(Task task)

- Adds all the main task details in the given form and saves it in the variabe ‘strsave’.

→ private static String encodeQueuedTasks(Task task)

- All the queued objects are added on to the main task object line once they are done

→ public static Boolean checkSlash(String userInput)

- Boolean function returns TRUE if there is a ‘/’ found in the userInput

# 2. Storage

Class which stores the data from the code into a text file and then updates the file when modifications are done to the file

EMPTY CONSTRUCTOR

**Methods**

→ public Storage(String filePath)

- Creates a filepath and the input is the file that all the data is being saved into

→ public void saveData(TaskList taskList)

- Writes the new data into the file

→ public TaskList loadData()

- Loads the data from the file that has previously been saved to it

→ public static Task loadTaskFromStorageString(String loadedInput[1])

- Takes in a line from savedData.txt and converts it into a Task Object
- The function parses the stored String and creates the Object, restoring all properties such as taskName and queuedTasks

# 3. Ui

The User Interface that will be at the forefront of this software and contains the basic commands to get the userInput and then outsources all the data to the other packages to get the task done

EMPTY CONSTRUCTOR

Variables
→ protected Scanner scanner;
→ protected Boolean exitRequest;
→ protected List<String> userInputHistory;
→ public static final String LOGO;
→ public static final String LINE;

**Methods**

→ public Ui()

- Initialises the scanner
- Creates a new array list to store the data
- Sets the exitRequest() boolean to false

→ public void initialise(TaskList taskList)

- Prints the welcome message

[1] Tasks are stored in the following String format:
<*TaskType*>_<*taskName*>/<*detalDesc*>_<*taskDesc*>####<*isDone*>-->><queuedTasks>

- Runs a while loop which calls the “interact” function continuously until the exit request is called upon

→ `public void interact(TaskList taskList)`
- Takes in the userInput
- Sends the userInput into the parser class
- Sends the tasklist into the execute method of the command class
- Sets the exitRequest method

→ `public String scanInput()`
- System will show “User: ” to take in the userInput
- Takes in the userInput
- Adds the userInput to the userInputHistory
- Returns the userInput

→ `public void exitUi()`
- Changes the exitRequest boolean to TRUE

→ `public static void printGoodbyeMsg()`
- Prints a line to end of the code
- Prints a line separator

→ `public static void executeCommand(CommandType commandType, String userInput)`
-

→ `public static void dukeSays(String stringX)`
- Method which prints out what is inputted in the method

→ `public static void printSeparator()`
- Prints a line

→ `public static void printWelcomeMsg()`
- Prints the welcome messages that has been set.

# Task Package

This package details all the functions relating to Tasks.

## Contents



## Task.java

Task.java is the main Task Object class from which all other Tasks (such as Deadline and Event extend from). Any method defined in this class will be inherited by all other subTasks.

Properties include:

**taskName -** The title of the Task.
**taskDetails -** The details of the Tasks (usually everything that comes after the backslash-space of the user input e.g. /xx_<taskDetails>).
**detailDesc -** The word attached to the backslash (/<detailDesc>_<taskDetails)
**TaskType -** The taskType Enum that describes the Task Object.
**isDone -** A boolean to check if the task is marked done.
**queuedTasks -** A TaskList (see below) containing all the queued tasks that are to be done once this Task is marked done.
**datetime -** A Date Object.

Methods include:

**Constructor:** Takes in user input as a string and removes the TaskType Enum (See below) before saving it all under the protected property: *taskName*

**getStatusIcon:** Returns a tick or cross depending on the *isDone* property.

**markDone:** Sets the *isDone* property to true.

**markNotDone:** Sets the *isDone* property to false.

**genTaskDesc:** returns a String that describes the Task entirely.

**genTypeAcronym:** returns the first letter of the TaskType enum

**recordTaskDetails:** takes in a user input and separates it based on this format

*<taskType>_<taskName>_/<detailDesc>_<taskDetails>*

## TaskList

TaskList is a class that essentially houses the ArrayList<Task> but with additional built in features.

Properties include:

**taskList -** ArrayList<Task>

Methods include:

**Constructor:** Does nothing

**deleteTaskByIndex:** Takes in a developer[2] index and removes it from the *taskList*

**addTask:** Takes in a Task Object to be added into the *taskList*

**findTasks:** Takes in a string and prints out all the Tasks containing that string

**printTaskByIndex:** Takes in a series of developer indexes and prints out the corresponding Task

**Static [3]createTask:** Creates a Task given a TaskType (see below) and a String containing the *taskName*[4] of the task onwards

**Static createTaskFromString:** Takes in a String from **STORAGE** and converts it into a Task Object

**printTasks:** Prints all the Tasks in the TaskList.

## TaskType

TaskType is the Enumeration that describe the various tasks such as Deadline and Event.

Methods include:

**Constructor:** Does nothing

**getNames:** Returns a String Array of all the different TaskType Enums such as Event.

---

[2] Developer index starts from 0 whereas user index starts from 1

[3] Static refers to methods that are not restricted to instantiated objects. They can be called by calling the Class directly.

[4] User Inputs for tasks are expected to follow this format <*TaskType*>_<*taskName*>_/<*detailDesc*>_<*taskDesc*>