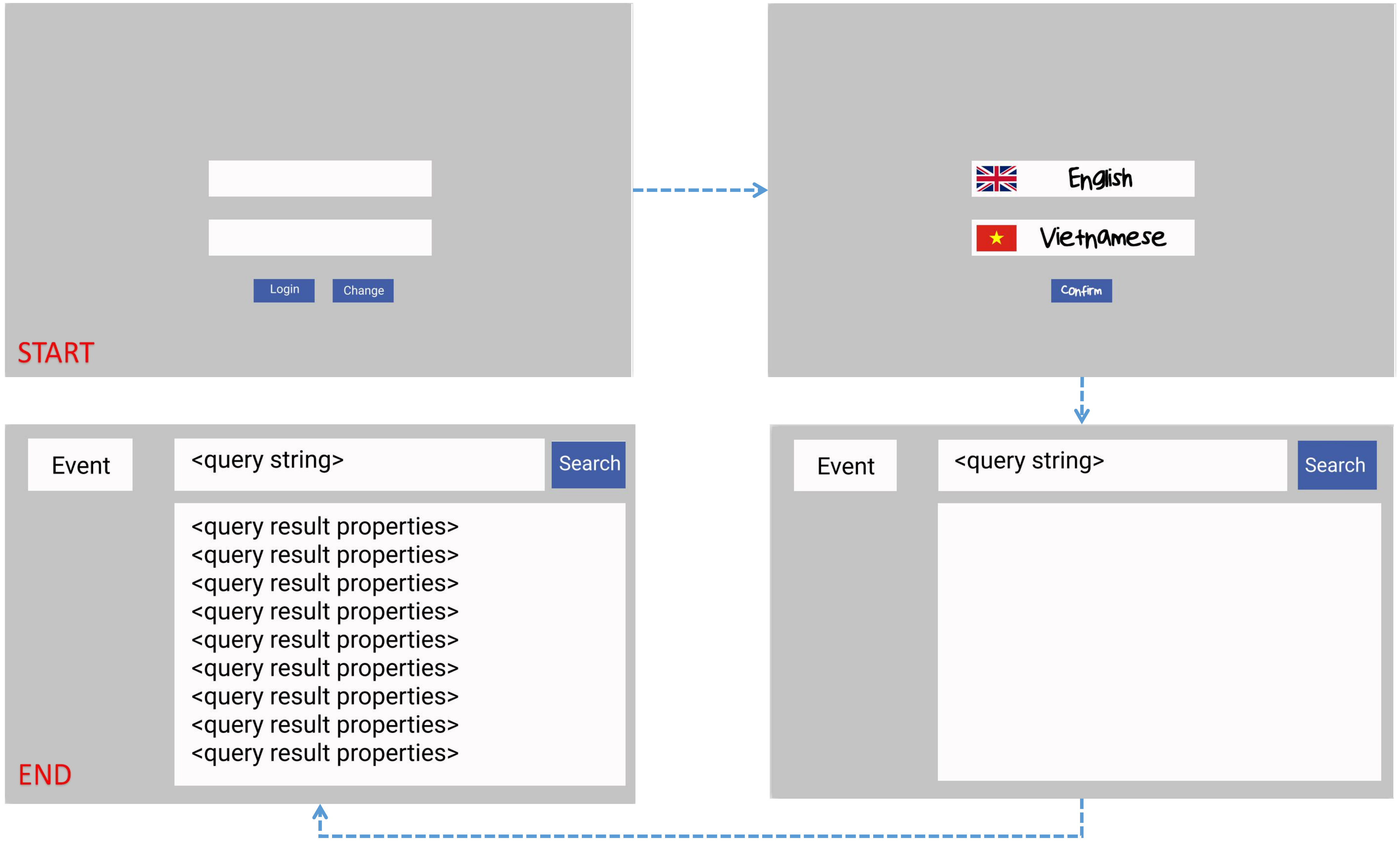
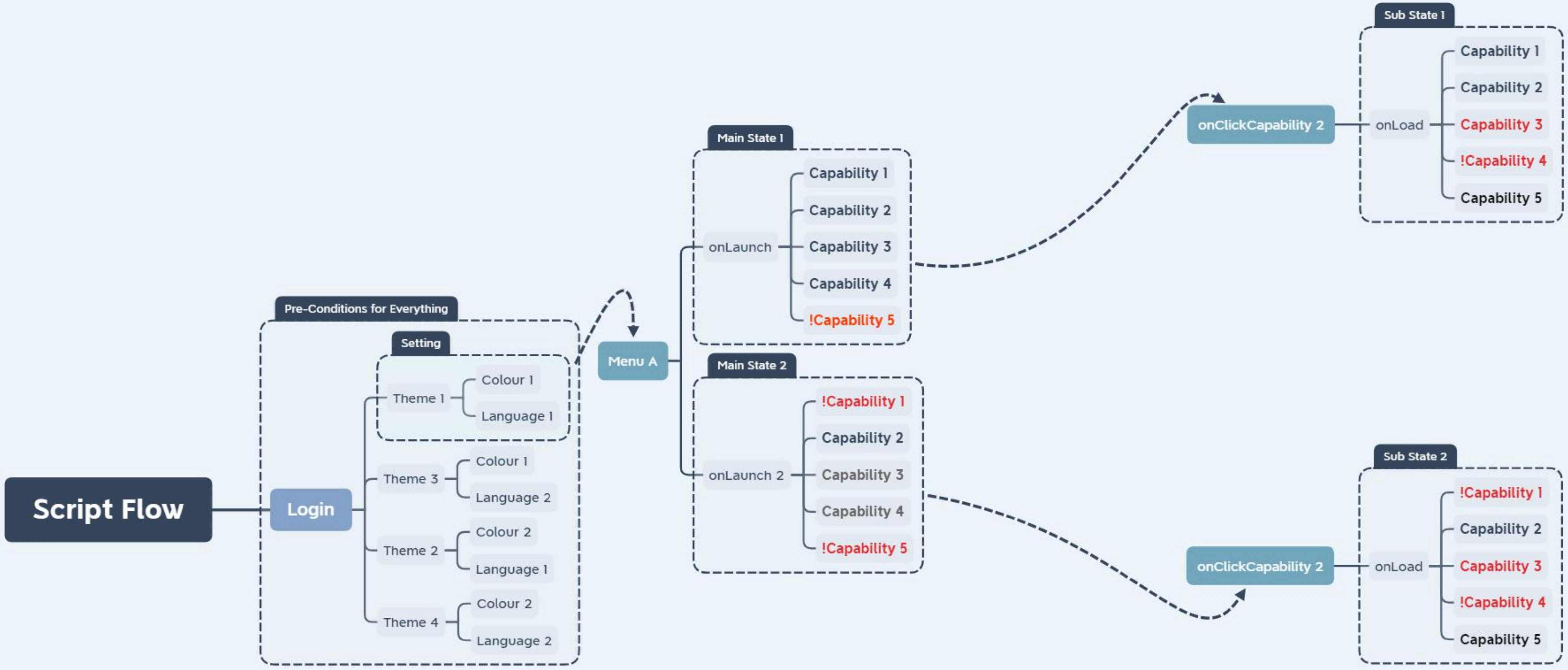

Login
Change
START
English
Vietnamese
Confirm
Event
<query string>
Search
Event
<query string>
Search
<query result properties>
<query result properties>
<query result properties>
<query result properties>
<query result properties>
<query result properties>
<query result properties>
<query result properties>
<query result properties>
END

Example Application: Does not related to any application by any means, do not re-distribute.

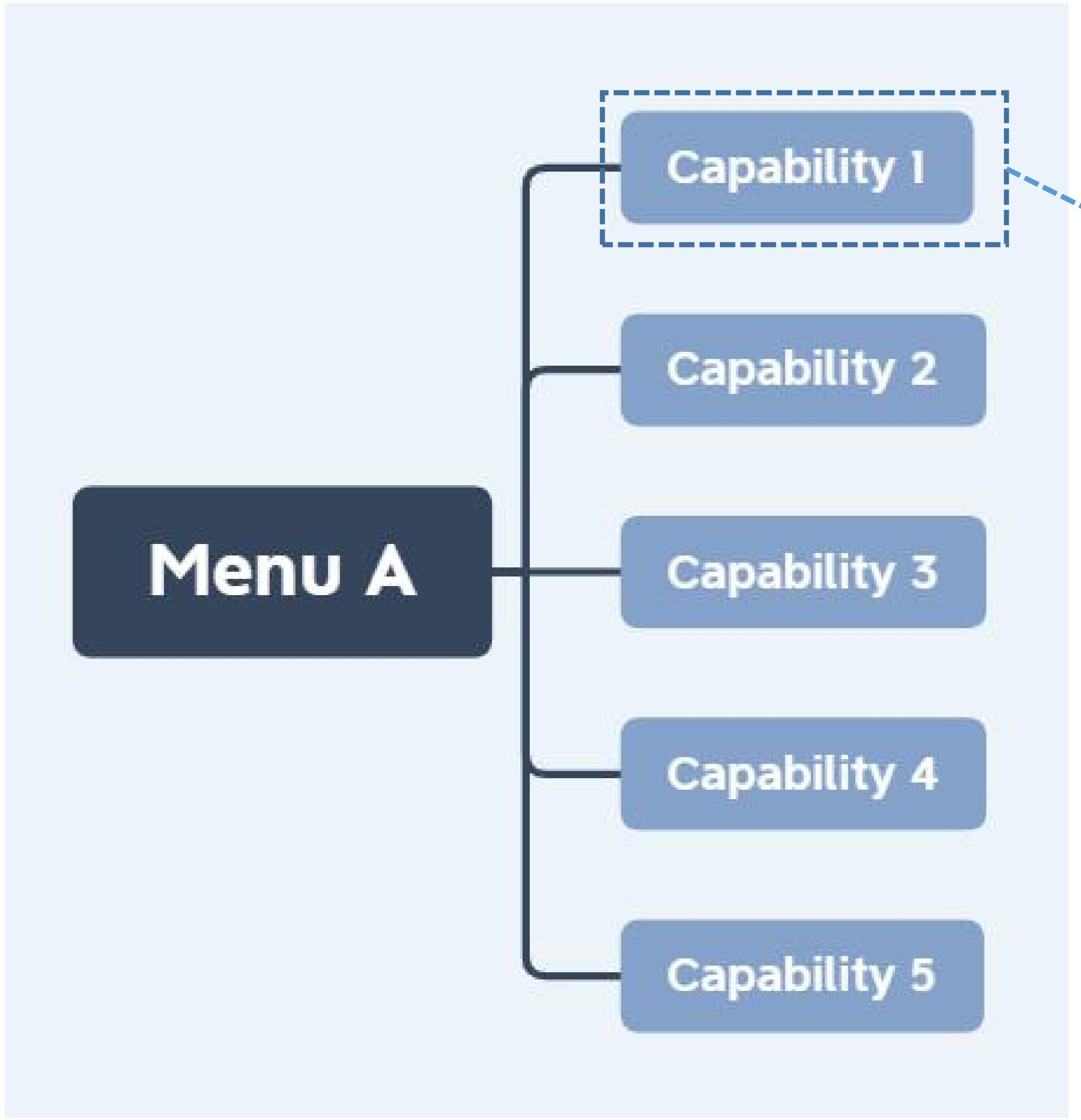


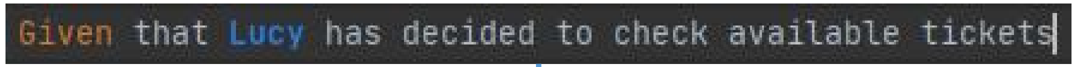
What does this mean?

On main state 1, user logged in with **Full Control** account, when get get to **Menu A**, only **Capability 5** is inaccessible for him/her.

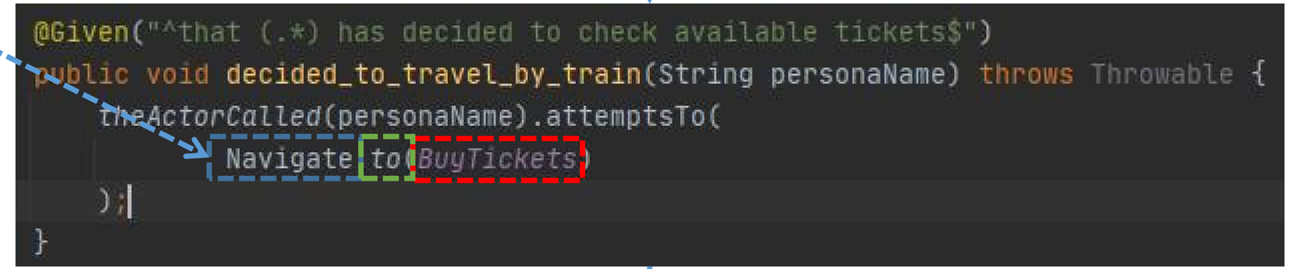
On main state 2, user logged in with **Standard** account, when he get to **Menu A**, **Capabilities 1 and 5** is inaccessible for him/her.

How to ensure that our test script cover all Capability?

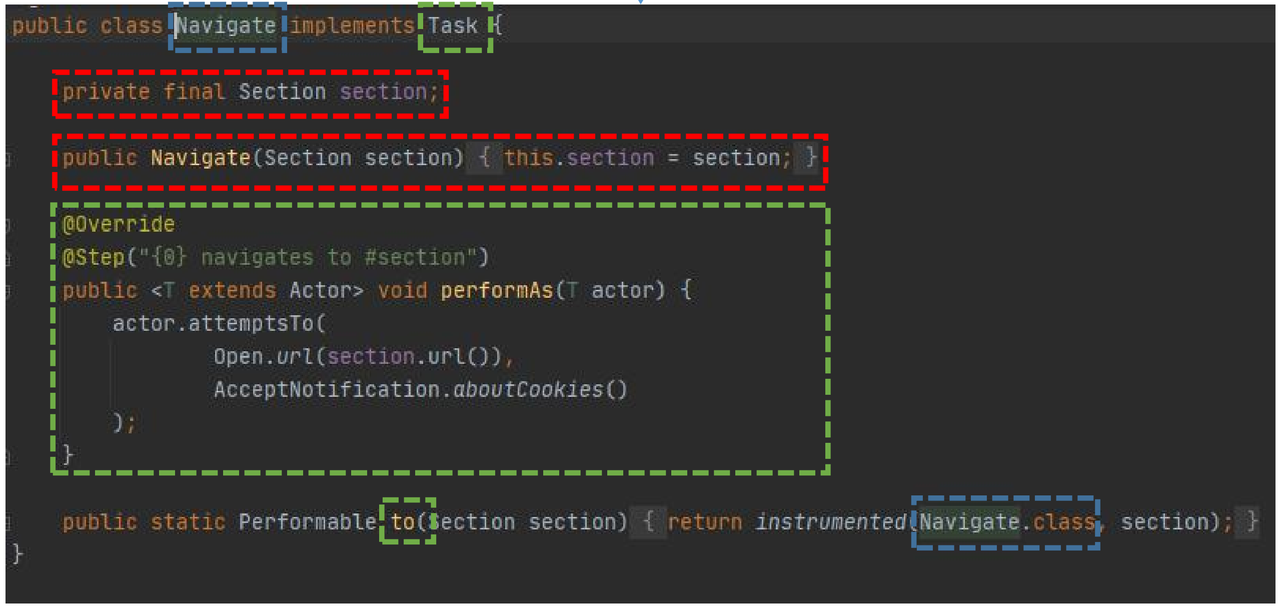




```
Given that Lucy has decided to check available tickets
```



```
@Given("^that (.*) has decided to check available tickets$")
public void decided_to_travel_by_train(String personaName) throws Throwable {
    theActorCalled(personaName).attemptsTo(
            Navigate.to(BuyTickets)
    );
}
```



```
public class Navigate implements Task {

    private final Section section;

    public Navigate(Section section) { this.section = section; }

    @Override
    @Step("{0} navigates to #section")
    public <T extends Actor> void performAs(T actor) {
        actor.attemptsTo(
                Open.url(section.url()),
                AcceptNotification.aboutCookies()
        );
    }

    public static Performable to(Section section) { return instrumented(Navigate.class, section); }
}
```

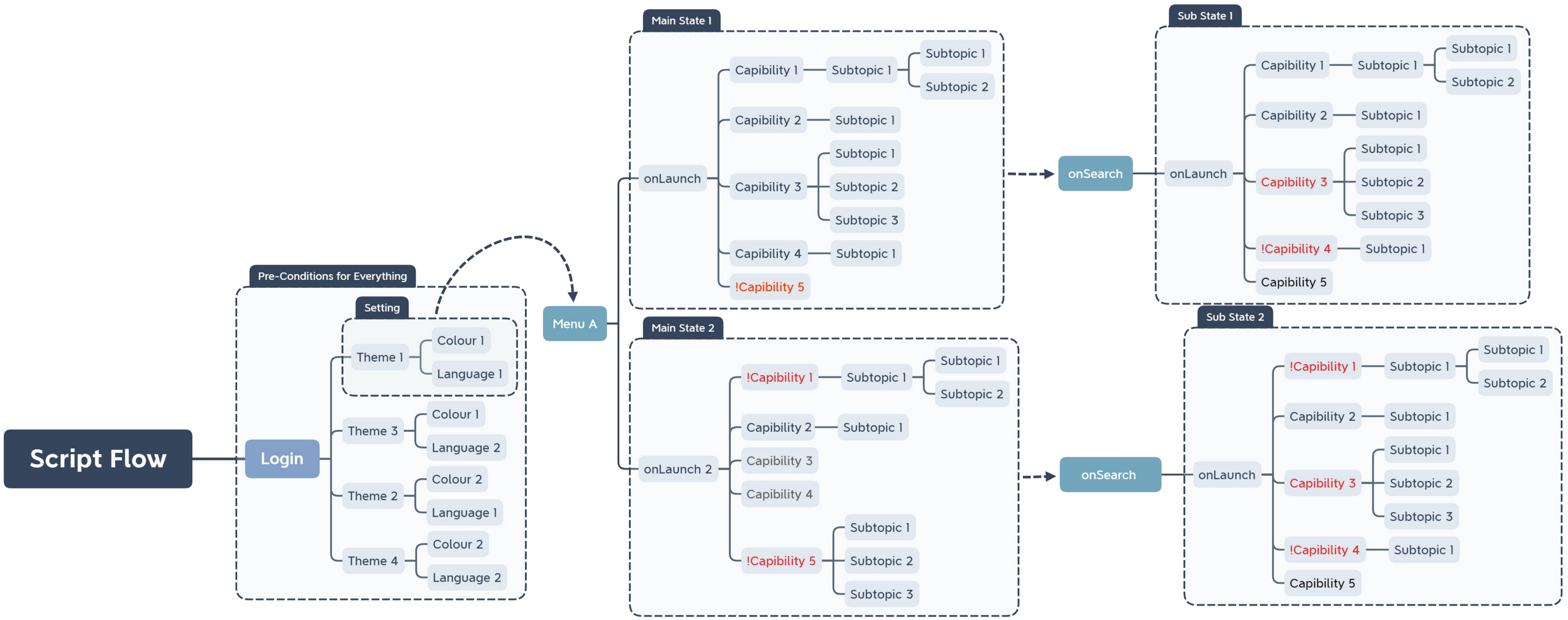
How to ensure that our test script cover all Capability?

Simply make Tasks class to cover all **Capabilities**, in the following example above, a class to cover 1 task.

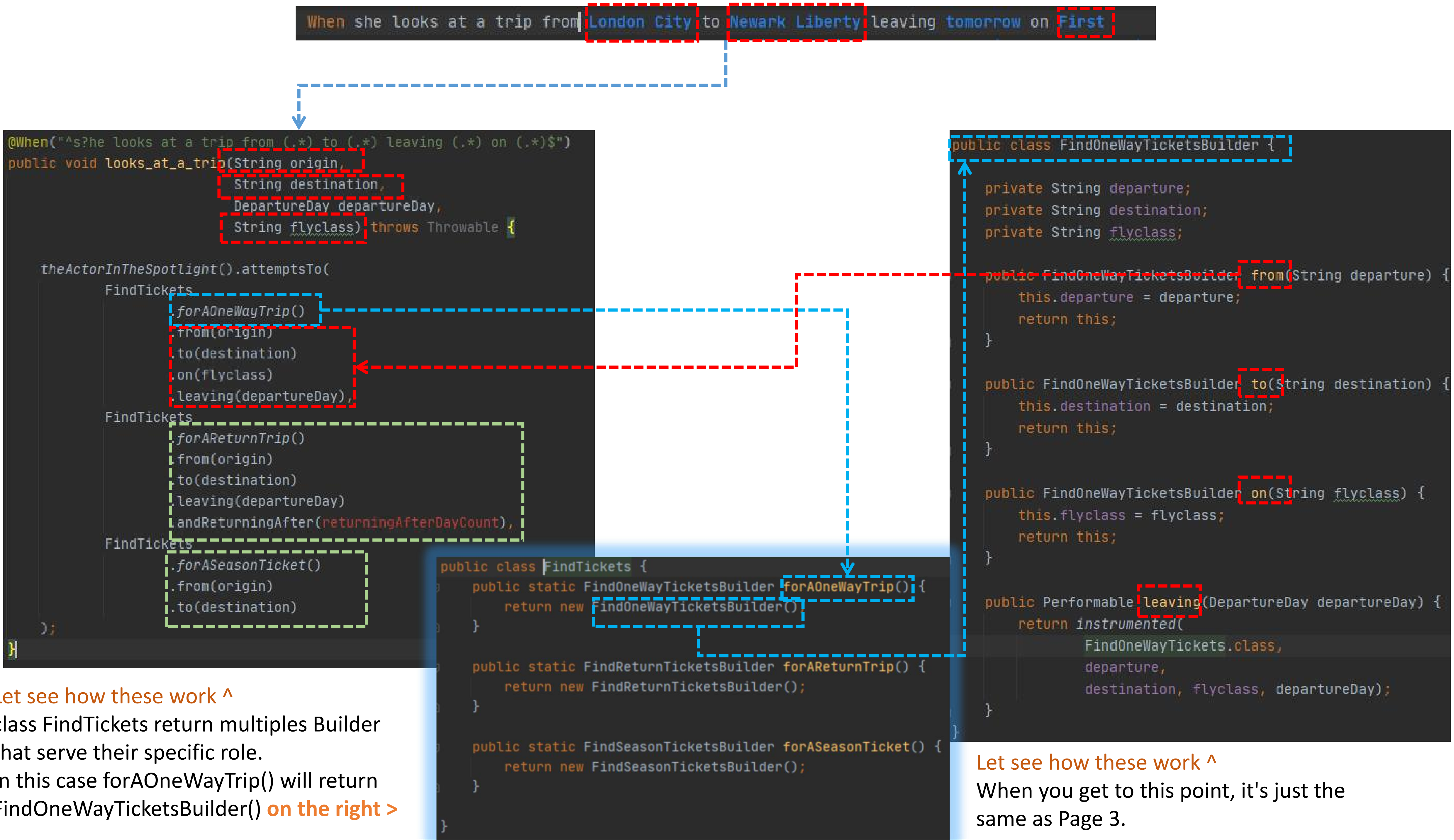
Blue group is our Capability Class that we named Navigate,

Green Group is a function that will perform actions using values from Red group,

Red group is a constructor to get value from various sources, in this case an **enum** file.



Let scale this up abit by adding sub Capabilities
How will our script handles these sub topic?
Simply Separate our **Builders** class by **Factory** them.



```
When she looks at a trip from London City to Newark Liberty leaving tomorrow on First
```

```
@When("^s?he looks at a trip from (.*) to (.*) leaving (.*) on (.*)$")
public void looks_at_a_trip(String origin,
                            String destination,
                            DepartureDay departureDay,
                            String flyclass) throws Throwable {

    theActorInTheSpotlight().attemptsTo(
            FindTickets
                    .forAOneWayTrip()
                    .from(origin)
                    .to(destination)
                    .on(flyclass)
                    .leaving(departureDay),
            FindTickets
                    .forAReturnTrip()
                    .from(origin)
                    .to(destination)
                    .leaving(departureDay)
                    .andReturningAfter(returningAfterDayCount),
            FindTickets
                    .forASeasonTicket()
                    .from(origin)
                    .to(destination)
    );
}
```

```
public class FindTickets {
    public static FindOneWayTicketsBuilder forAOneWayTrip() {
        return new FindOneWayTicketsBuilder();
    }

    public static FindReturnTicketsBuilder forAReturnTrip() {
        return new FindReturnTicketsBuilder();
    }

    public static FindSeasonTicketsBuilder forASeasonTicket() {
        return new FindSeasonTicketsBuilder();
    }
}
```

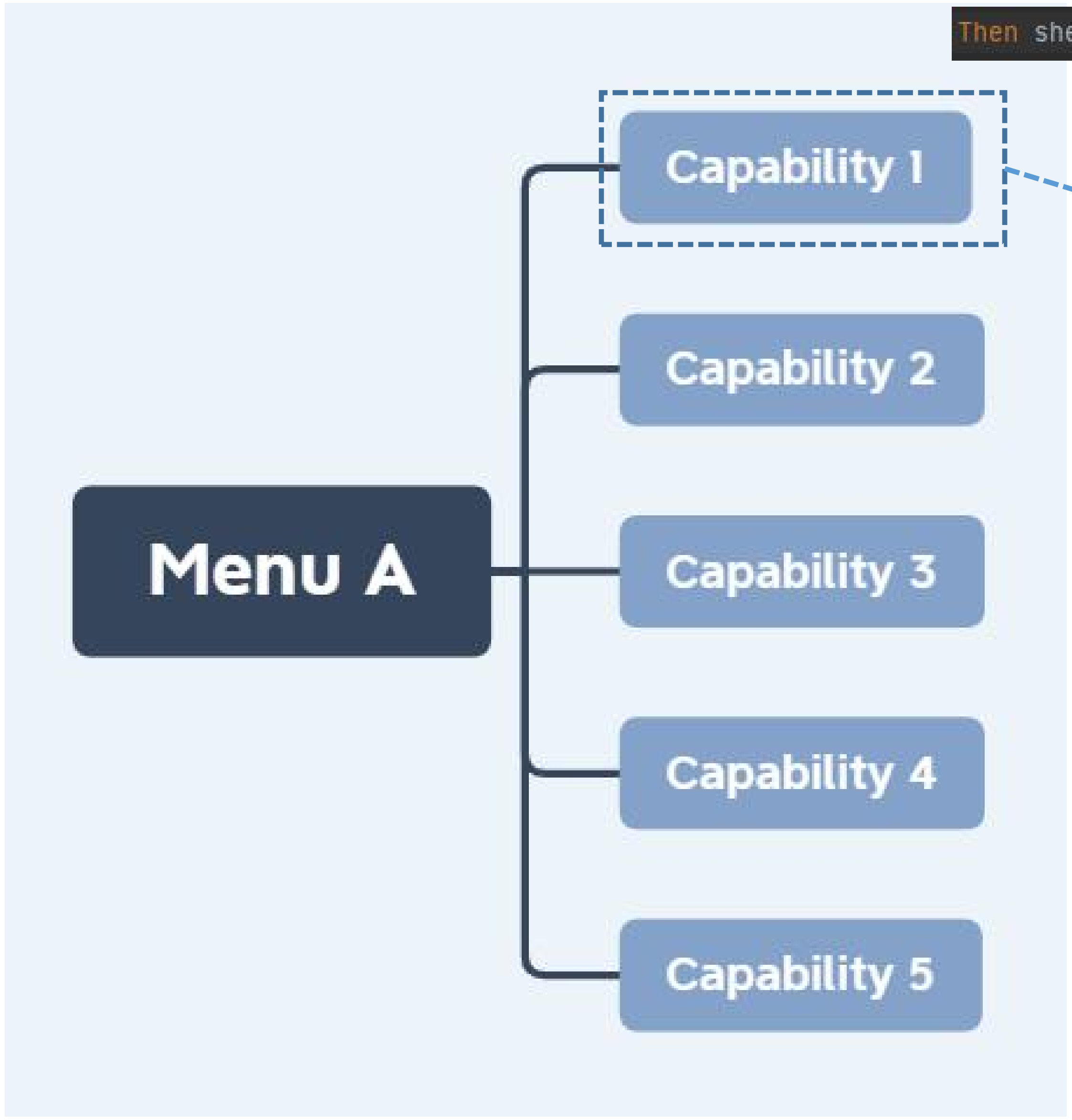
```
public class FindOneWayTicketsBuilder {

    private String departure;
    private String destination;
    private String flyclass;

    public FindOneWayTicketsBuilder from(String departure) {
        this.departure = departure;
        return this;
    }

    public FindOneWayTicketsBuilder to(String destination) {
        this.destination = destination;
        return this;
    }

    public FindOneWayTicketsBuilder on(String flyclass) {
        this.flyclass = flyclass;
        return this;
    }

    public Performable leaving(DepartureDay departureDay) {
        return instrumented(
                FindOneWayTickets.class,
                departure,
                destination, flyclass, departureDay);
    }
}
```

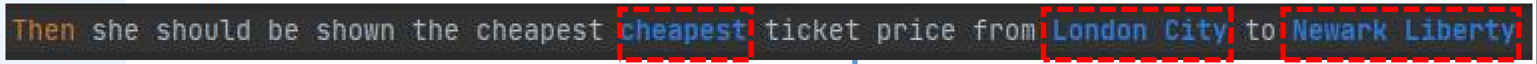
Let see how these work ^

class FindTickets return multiples Builder that serve their specific role.
In this case forAOneWayTrip() will return FindOneWayTicketsBuilder() **on the right >**

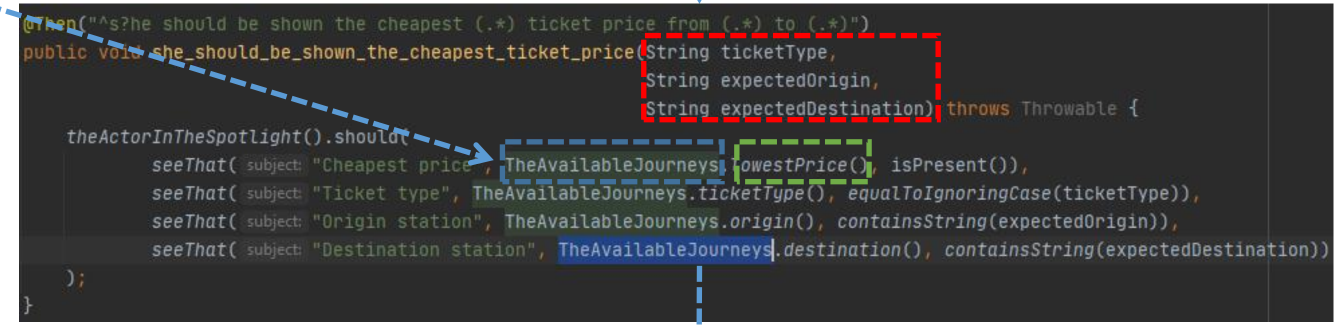
Let see how these work ^

When you get to this point, it's just the same as Page 3.

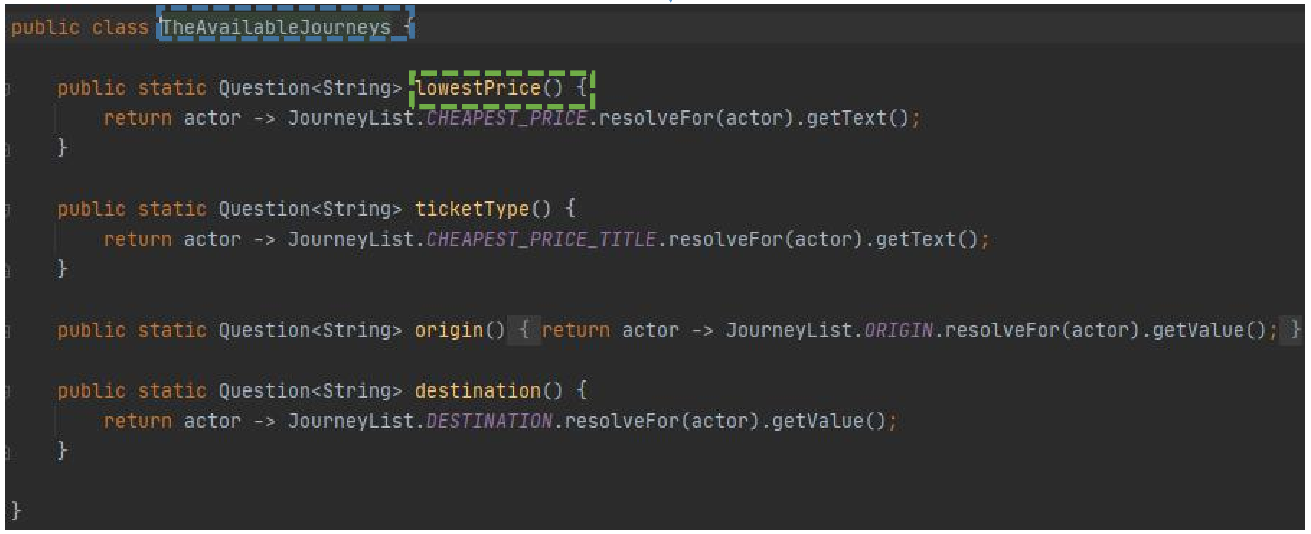




```
Then she should be shown the cheapest cheapest ticket price from London City to Newark Liberty
```



```
@Then("^s?he should be shown the cheapest (.*) ticket price from (.*) to (.*)")
public void she_should_be_shown_the_cheapest_ticket_price(String ticketType,
                                                          String expectedOrigin,
                                                          String expectedDestination) throws Throwable {

    theActorInTheSpotlight().should(
            seeThat( subject: "Cheapest price", TheAvailableJourneys.lowestPrice(), isPresent()),
            seeThat( subject: "Ticket type", TheAvailableJourneys.ticketType(), equalToIgnoringCase(ticketType)),
            seeThat( subject: "Origin station", TheAvailableJourneys.origin(), containsString(expectedOrigin)),
            seeThat( subject: "Destination station", TheAvailableJourneys.destination(), containsString(expectedDestination))
    );
}
```



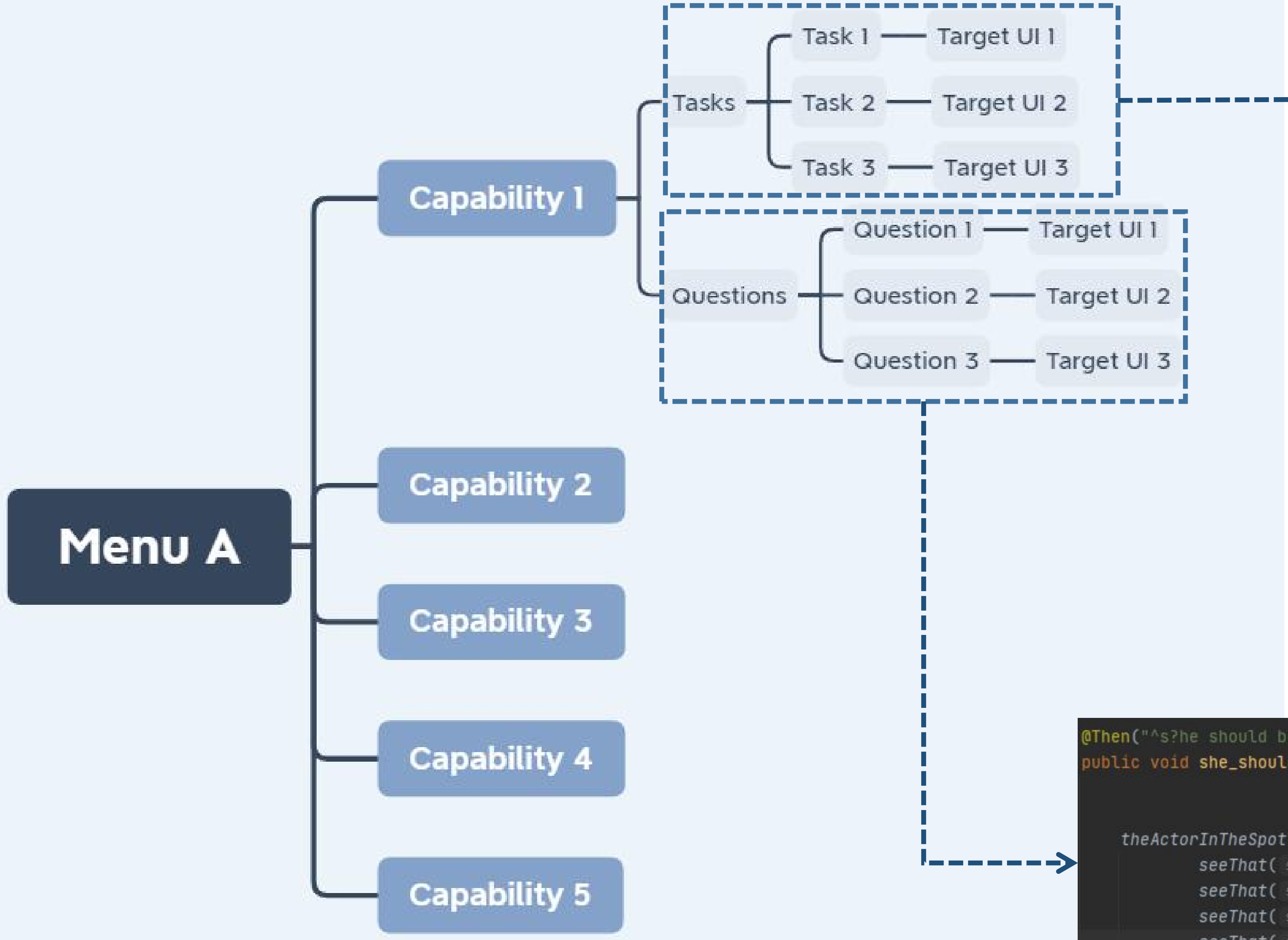
```
public class TheAvailableJourneys {

    public static Question<String> lowestPrice() {
        return actor -> JourneyList.CHEAPEST_PRICE.resolveFor(actor).getText();
    }

    public static Question<String> ticketType() {
        return actor -> JourneyList.CHEAPEST_PRICE_TITLE.resolveFor(actor).getText();
    }

    public static Question<String> origin() { return actor -> JourneyList.ORIGIN.resolveFor(actor).getValue(); }

    public static Question<String> destination() {
        return actor -> JourneyList.DESTINATION.resolveFor(actor).getValue();
    }

}
```

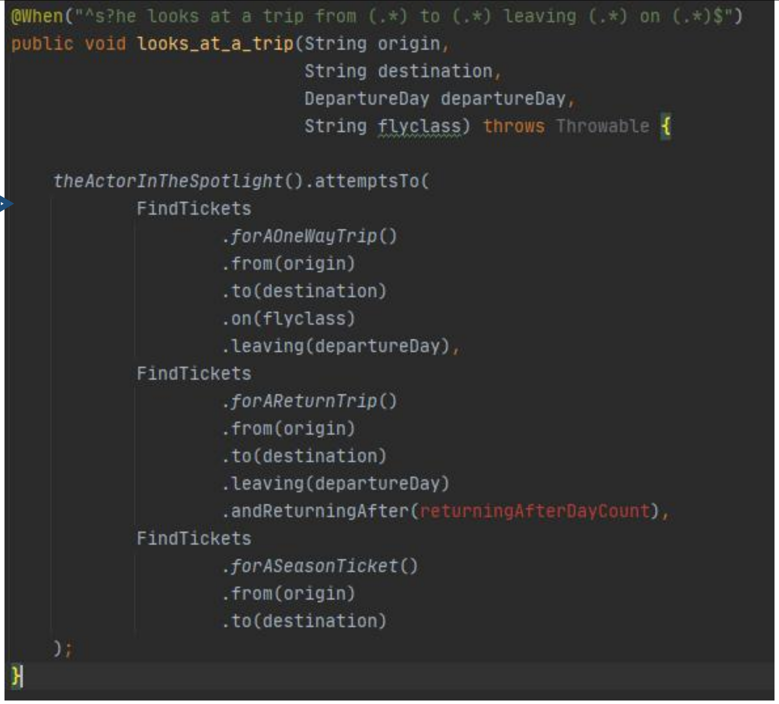
How to prove that the Capibility work like how we expect it to? **Use Question Class**

Blue group is our Question Class that that share the same ancestry,

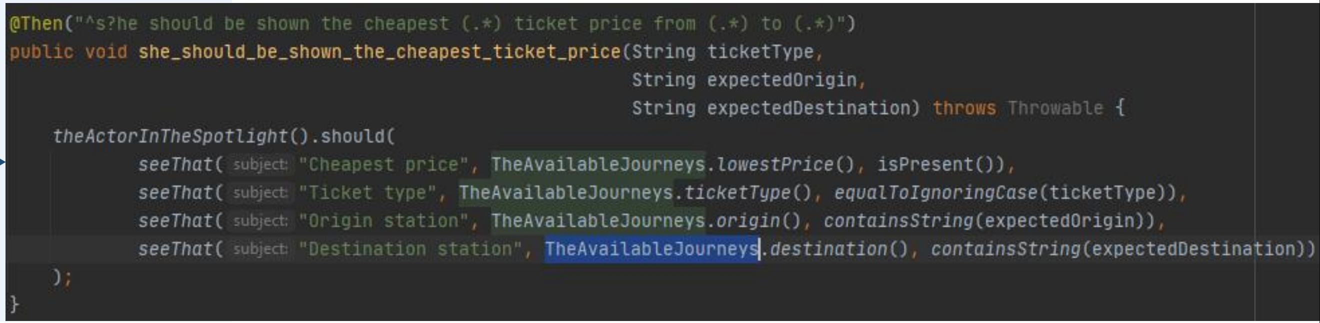
Green Group is a function that will return a value from the web driver to compare with red group ie String, Int, Boolean, List, ect

Red group is how we parse values to compare with value we can read from the web driver.





```
@When("^s?he looks at a trip from (.*) to (.*) leaving (.*) on (.*)$")
public void looks_at_a_trip(String origin,
                            String destination,
                            DepartureDay departureDay,
                            String flyclass) throws Throwable {

    theActorInTheSpotlight().attemptsTo(
            FindTickets
                    .forAOneWayTrip()
                    .from(origin)
                    .to(destination)
                    .on(flyclass)
                    .leaving(departureDay),
            FindTickets
                    .forAReturnTrip()
                    .from(origin)
                    .to(destination)
                    .leaving(departureDay)
                    .andReturningAfter(returningAfterDayCount),
            FindTickets
                    .forASeasonTicket()
                    .from(origin)
                    .to(destination)
    );
}
```



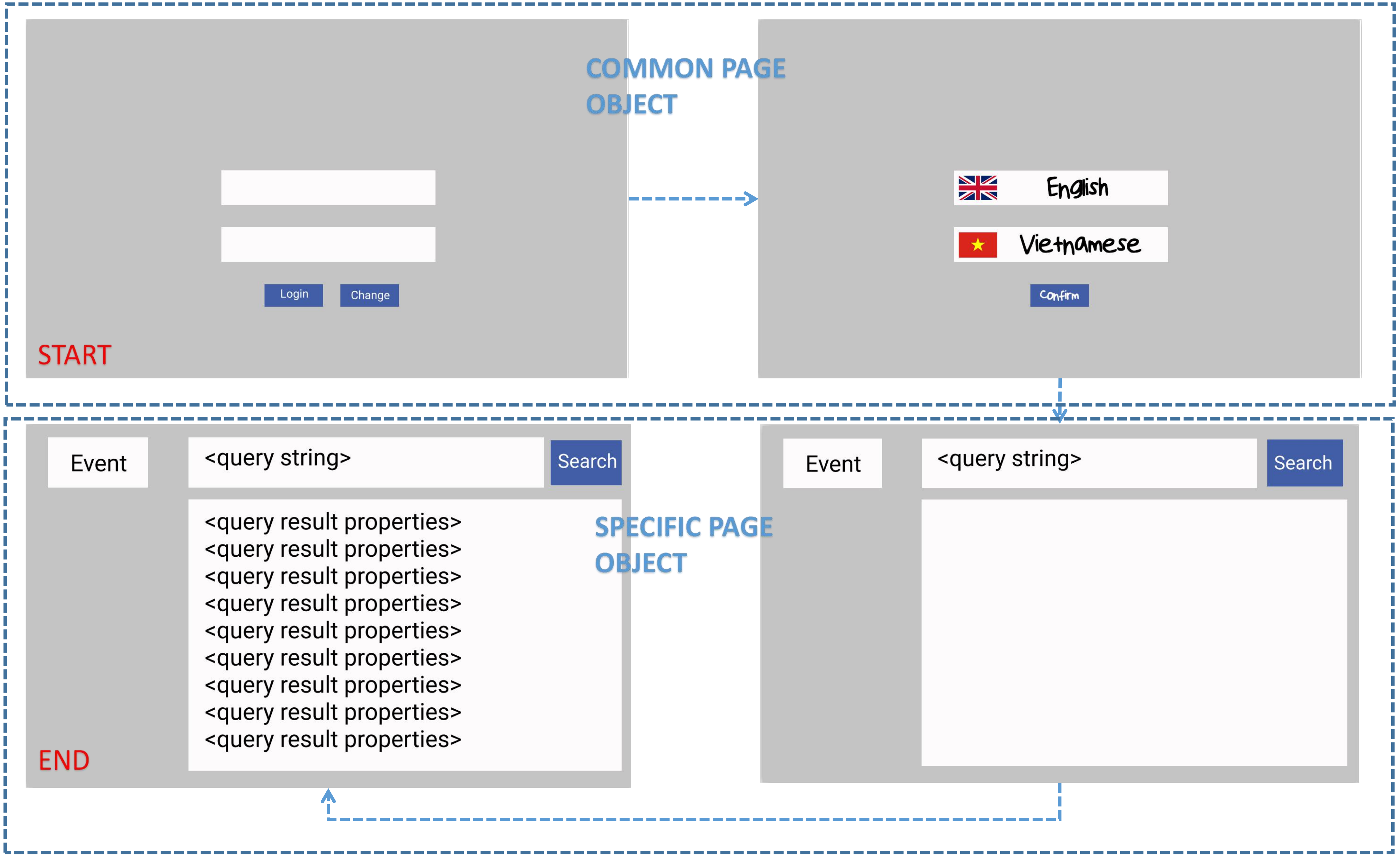
```
@Then("^s?he should be shown the cheapest (.*) ticket price from (.*) to (.*)")
public void she_should_be_shown_the_cheapest_ticket_price(String ticketType,
                                                          String expectedOrigin,
                                                          String expectedDestination) throws Throwable {
    theActorInTheSpotlight().should(
            seeThat( subject: "Cheapest price", TheAvailableJourneys.lowestPrice(), isPresent()),
            seeThat( subject: "Ticket type", TheAvailableJourneys.ticketType(), equalToIgnoringCase(ticketType)),
            seeThat( subject: "Origin station", TheAvailableJourneys.origin(), containsString(expectedOrigin)),
            seeThat( subject: "Destination station", TheAvailableJourneys.destination(), containsString(expectedDestination))
    );
}
```

Sum-up

Tasks created based on the **Capability**, its name will varies from the type, name it as first letter of each word in upper case for example: **F**ind**T**ickets.

Questions created based on the **Capability**, its name will varies from the type, name it as first letter of each word in upper case for example: **The****A**vailable**X** .

How will we save the locator in our current project?


COMMON PAGE OBJECT
Login
Change
START
English
Vietnamese
Confirm
Event
<query string>
Search
<query result properties>
<query result properties>
<query result properties>
<query result properties>
<query result properties>
<query result properties>
<query result properties>
<query result properties>
<query result properties>
END
SPECIFIC PAGE OBJECT
Event
<query string>
Search

Example Application: Does not related to any application by any means, do not re-distribute.