Oracle MOOC: Oracle Intelligent Bots

Session 2

Define the Chatbot Flow Using BotML

In this lab, you will add BotML to your MasterBot. The code will support each of your intents. Specifically, you will set up the Balances and Send Money intent variables and then add all the states that are needed to complete their actions. When you're done, you will test both of those intents to make sure that they work as expected. After that, you'll set up the variable and a starting point in the flow for the Track Spending intent. You'll add a state for this intent as well, but it will be just a shell. You'll complete it in the next lab.

Before You Begin

You need the following files for this lab, which are located in the `/labfiles/code` directory:

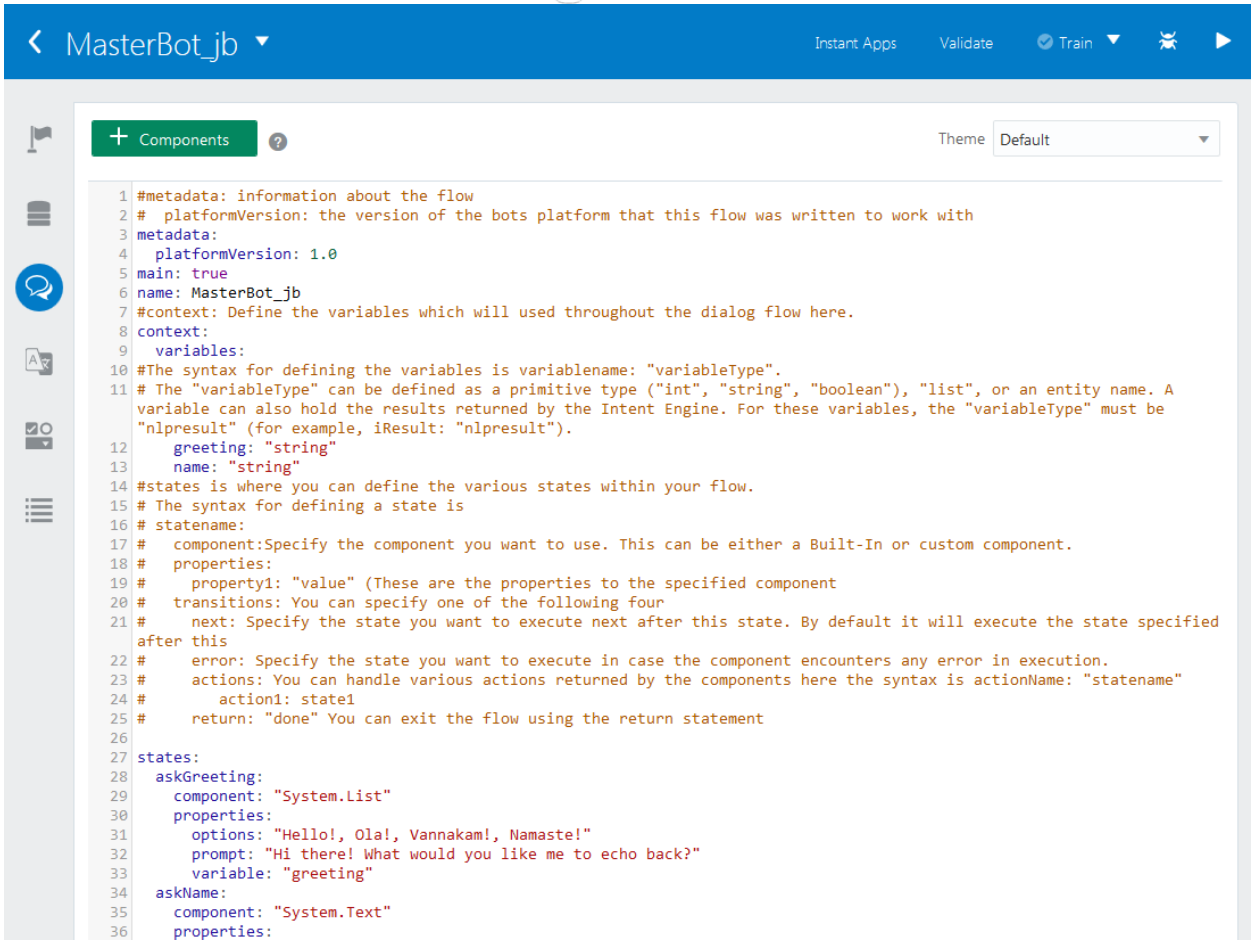
- `FirstBotYAML.txt`
- `startPayment.txt`
- `processPayment.txt`
- `doPayment.txt`
- `startTrackSpending.txt`

Step 1: Include the Code that Supports the Balances Intent

In this section, you add code to the flow of your chatbot that supports the Balances intent. You used this code in an earlier lab, but here we'll examine its components and use it as an example of how you should set up other components.

1. Open your MasterBot_xx chatbot and click the **Flows** icon in the left navbar. You will see the default BotML "Hello" code in the editor. You will not need any of it, so delete it.

Oracle MOOC: Oracle Intelligent Bots



2. If you haven't already done so, locate the `FirstBotYAML.txt` in the `/labfiles/code` directory, open it, copy its contents into the editor, and then click **Validate**.

Oracle MOOC: Oracle Intelligent Bots

```yaml
metadata:
  platformVersion: "1.0"
main: true
name: "FinancialBotMainFlow"
context:
  variables:
    accountType: "AccountType"
    iResult: "nlpresult"
states:
  intent:
    component: "System.Intent"
    properties:
      variable: "iResult"
      confidenceThreshold: 0.4
    transitions:
      actions:
        Balances: "startBalances"
        unresolvedIntent: "unresolved"
  startBalances:
    component: "System.SetVariable"
    properties:
      variable: "accountType"
      value: "${iResult.value.entityMatches['AccountType'][0]}"
    transitions: {}
  askBalancesAccountType:
    component: "System.List"
    properties:
      options: "${accountType.type.enumValues}"
      prompt: "For which account do you want your balance?"
      variable: "accountType"
    transitions: {}
  printBalance:
    component: "System.Output"
    properties:
      text: "Balance for ${accountType.value} is $500"
    transitions:
      return: "printBalance"
  unresolved:
    component: "System.Output"
    properties:
      text: "Unable to resolve intent!"
    transitions:
      return: "unresolved"
```

3. Let's look at the code we have and dissect what it represents.
   - Here we see a variety of sections to the code: the header followed by the declaration of content variables and then the intent states.
   - The `accountType` variable is used for recording and displaying the account for which the balance is requested. It's used in the

Oracle MOOC: Oracle Intelligent Bots

`startBalances` state to check if what the user has typed in matches any of the values included in the entity definition. It is also used in the `askBalancesAccountType` state to store the value that's entered if the account type is not specified in the `startBalances` state. Finally, `accountType` is used in the `printBalance` state when the balance is displayed to the user.

- There are five states defined. The `intent` state is the result of intent classification and entity resolution that's provided by the Intent Engine as the `nlpresult` (of which `iResult` is a type). In other words, this variable (`iResult`) holds the result of the Intent Engine (that is, the intent and entity resolution) from the input text provided by the user. The `actions` show a `startBalances` state which implements the Balances intent. There is also another state listed that's used when the intent cannot be resolved: at the end of the flow, notice the `unresolved` state, which is reached if no other state is fulfilled.
- Below the `intent` state are all the other states that the chatbot uses. In our case, the `startBalances` state is the entry point for Balances, regardless if the `accountType` variable is set in `StartBalances`, or if the `nlpresult` (that is, the `iResult` variable) provides this value from parsing user input. In this case, `askBalancesAccountType` does not attempt to set the variable because it's already been set (and as a result, the list of options will not be displayed). If the value isn't set in the `nlpresult`, however, the flow attempts to set it using list of options. If an account type is not mentioned in the user's message, then a `System.List` component displays the valid account types as options. Once an account type is set, the control of the flow drops down to the `printBalance` state, where a `System.Output` component is used to display what the account type actually is and the balance amount. The flow moves to the final state (`unresolved`) if the intent cannot be resolved.

Oracle MOOC: Oracle Intelligent Bots

Header

Context Variables

Intent States

Start Balance

Ask Balance Account Type

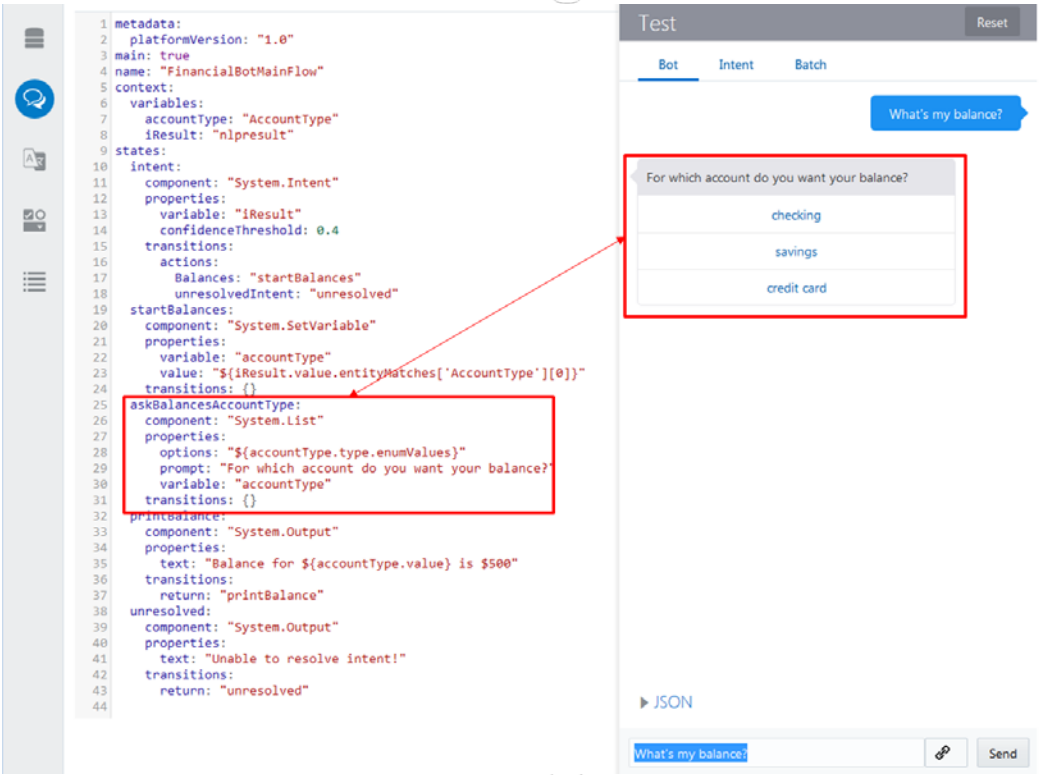
Print Balance

Unresolved

```
1  metadata:
2    platformVersion: "1.0"
3  main: true
4  name: "FinancialBotMainFlow"
5  context:
6    variables:
7      accountType: "AccountType"
8      iResult: "nlpresult"
9  states:
10   intent:
11     component: "System.Intent"
12     properties:
13       variable: "iResult"
14       confidenceThreshold: 0.4
15     transitions:
16       actions:
17         Balances: "startBalances"
18         unresolvedIntent: "unresolved"
19   startBalances:
20     component: "System.SetVariable"
21     properties:
22       variable: "accountType"
23       value: "${iResult.value.entityMatches['AccountType'][0]}"
24     transitions: {}
25   askBalancesAccountType:
26     component: "System.List"
27     properties:
28       options: "${accountType.type.enumValues}"
29       prompt: "For which account do you want your balance?"
30       variable: "accountType"
31     transitions: {}
32   printBalance:
33     component: "System.Output"
34     properties:
35       text: "Balance for ${accountType.value} is $500"
36     transitions:
37       return: "printBalance"
38   unresolved:
39     component: "System.Output"
40     properties:
41       text: "Unable to resolve intent!"
42     transitions:
43       return: "unresolved"
```

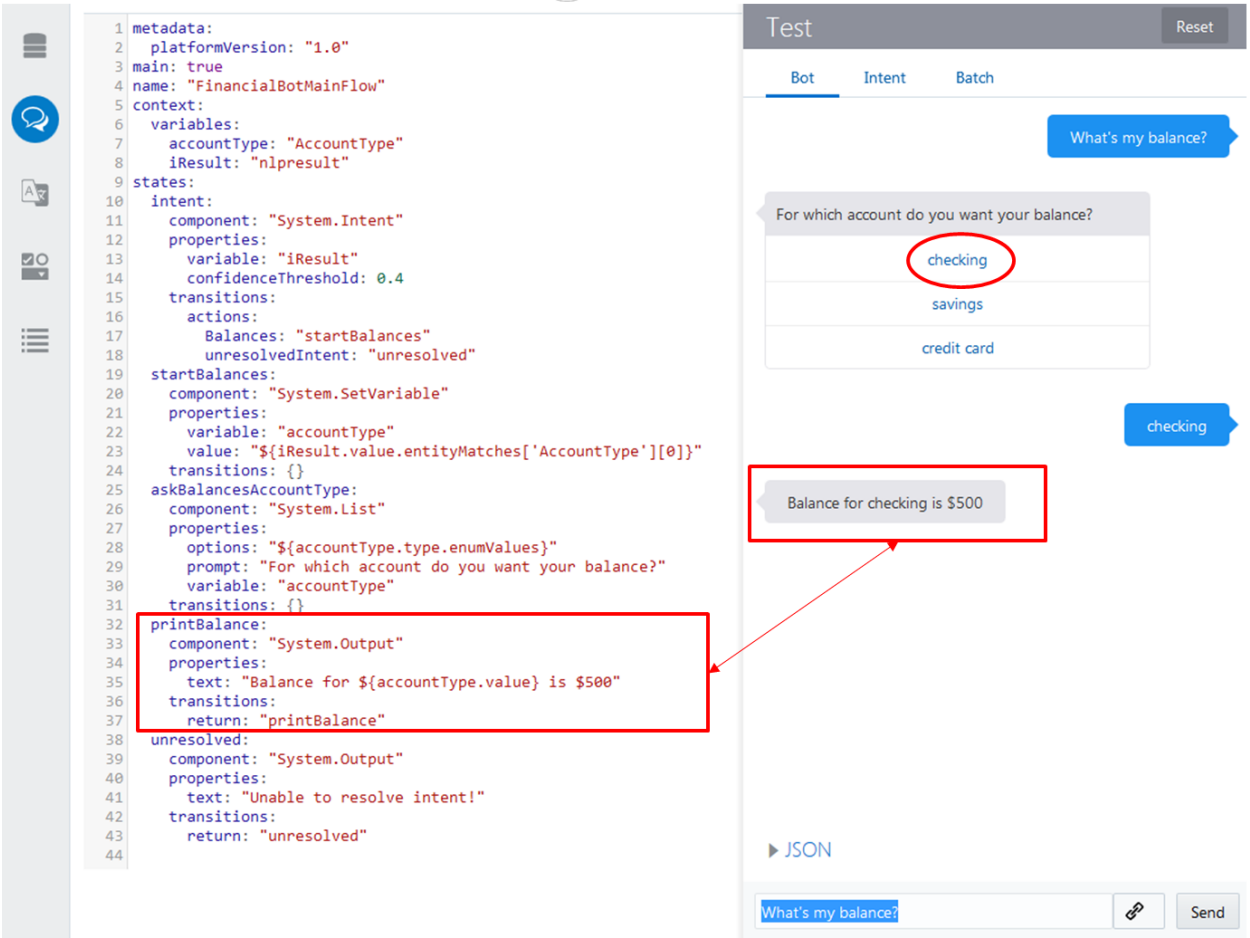
4. To test the flow code, click the **Play** button in the upper right, select the Bot tab in the Tester.
5. Enter *What's my balance?* in the Message area, and then click **Send**. You should see a list of all the accounts included in the `System.List` component.
6. Notice the prompt and the list of accounts. They originate from the `askBalanceAccountType` state.

Oracle MOOC: Oracle Intelligent Bots



7. Next, select an account. The return should display the `System.Output` component from the `printBalance` state and show you the amount in that account. Notice that the displayed value of $500 is hard-coded in the `System.Output` component (`“Balance for ${accountType.value} is $500”`).

Oracle MOOC: Oracle Intelligent Bots



8. Click **Reset** and then try out some other messages, including some with the account in the message text to see how your chatbot responds.

In the next section, you'll add some code for the Send Money intent.

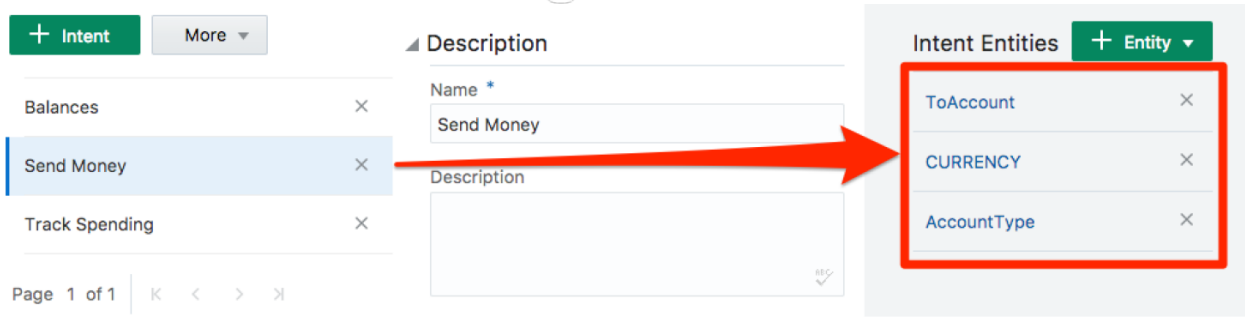
Step 2: Add code to support the Send Money intent

In this section, you will include the variables, components, and states that make the Send Money intent work.

> **Important**: To avoid any odd characters getting copied into the flow, type in the values for the next couple of steps instead of using copy and paste.

1. To send money, we'll need three pieces of information: the account the money is coming from, the account, or person, that the money is going to, and the amount of money to be transferred. We know these things because they are the entities used by the Send Money intent.

Oracle MOOC: Oracle Intelligent Bots



2. We already have a variable in place for the account where the money should come from when retrieving the balance: `accountType`. We just need two more: one for the account that the money goes to and one for the amount of money to be transferred.

   In the variables section under the `accountType` variable, add the following two variables. Be sure to indent them to the same level as the other variables.

   `toAccount: "ToAccount"`

   `paymentAmount: "CURRENCY"`

```
metadata:
  platformVersion: "1.0"
main: true
name: "FinancialBotMainFlow"
context:
  variables:
    accountType: "AccountType"
    toAccount: "ToAccount"
    paymentAmount: "CURRENCY"
    ikesult: "hipresult"
states:
  intent:
```

3. Next, we need to define an entry point for the Send Money intent.

   First, locate the `intent` state. In its `transitions` section, just after the `Balances` action, add the following entry point to the Send Money intent:

   `Send Money: "startPayments"`

Oracle MOOC: Oracle Intelligent Bots

```
11 states:
12   intent:
13     component: "System.Intent"
14     properties:
15       variable: "iResult"
16       confidenceThreshold: 0.4
17     transitions:
18       actions:
19         Balances: "startBalances"
20         Send Money: "startPayments"
21         unresolvedIntent: "unresolved"
22   startBalances:
23     component: "System.SetVariable"
```

4. So we now have variables and a starting point for the intent. To complete the Send Money intent, we need a few things to happen:

   - Determine the account that the money should go to.
   - Determine the account that the money should come from.
   - Determine the amount to send.
   - Send a notification that the payment has been made.

   Before we add the above states, we need to include the `startPayments` state along with another state to detect if the *to* account value is included in the initial message. In the code, find the `printBalance` state and then add the state to start the payment right after it. You enter this code manually, or copy it from the `startPayment.txt` file, which is located in the `/labfiles/code/` directory. Click **Validate** when you're done.

   Important: Make sure that the alignment is set just as it is in the following image. Otherwise, you will get an error message when you validate the flow.

Oracle MOOC: Oracle Intelligent Bots

```
35    printBalance:
36      component: "System.Output"
37      properties:
38        text: "Balance for ${accountType.value} is $500"
39      transitions:
40        return: "printBalance"
41    startPayments:
42      component: "System.SetVariable"
43      properties:
44        variable: "accountType"
45        value: "${iResult.value.entityMatches['AccountType'][0]}"
46      transitions: {}
47    resolveToAccount:
48      component: "System.SetVariable"
49      properties:
50        variable: "toAccount"
51        value: "${iResult.value.entityMatches['ToAccount'][0]}"
52      transitions: {}
53    unresolved:
54      component: "System.Output"
55      properties:
56        text: "Unable to resolve intent!"
57      transitions:
58        return: "unresolved"
```

5. Next, we will add the four states that determine what account the money should go to, where it should come from, and the amount itself. Let's describe them first before adding them.

   The first state, `askFromAccountType`, operates just like it did in the Balances intent: a `System.List` component is used to display the options along with a prompt asking which account to take the payment from. The options are provided by the values assigned to the `accountType` entity. The value selected in the message is then placed in the `accountType` variable.

```
askFromAccountType:
  component: "System.List"
  properties:
    options: "${accountType.type.enumValues}"
    prompt: "From which account do you want to make a payment?"
    variable: "accountType"
  transitions: {}
```

   The second state, `askToAccount`, also uses a `System.List` component to display the options for the money's recipient along with a prompt. Like with the `askFromAccountType` state, the options are provided by the values assigned to

Oracle MOOC: Oracle Intelligent Bots

the `toAccount` entity. The value selected in the message is then placed in the `toAccount` variable.

```
askToAccount:
  component: "System.List"
  properties:
    options: "${toAccount.type.enumValues}"
    prompt: "To which account do you want to make a payment?"
    variable: "toAccount"
  transitions: {}
```

The third state, `resolvePaymentAmount`, prompts the user to enter a value which is then placed in the `paymentAmount` variable.

```
resolvePaymentAmount:
  component: "System.SetVariable"
  properties:
    variable: "paymentAmount"
    value: "${iResult.value.entityMatches['CURRENCY'][0]}"
  transitions: {}
```

The fourth state, `askPaymentAmount`, prompts the user for the payment amount.

```
askPaymentAmount:
  component: "System.Text"
  properties:
    prompt: "What's the payment amount?"
    variable: "paymentAmount"
  transitions: {}
```

Add these four states below the `resolveToAccount` state. Again, be mindful of the indentation and click **Validate**.

You can find the code for these states in the `processPayment.txt` file, which is located in the `labfiles/code` directory.

Oracle MOOC: Oracle Intelligent Bots

```
47  resolveToAccount:
48    component: "System.SetVariable"
49    properties:
50      variable: "toAccount"
51      value: "${iResult.value.entityMatches['ToAccount'][0]}"
52    transitions: {}
53  askFromAccountType:
54    component: "System.List"
55    properties:
56      options: "${accountType.type.enumValues}"
57      prompt: "From which account do you want to make a payment?"
58      variable: "accountType"
59    transitions: {}
60  askToAccount:
61    component: "System.List"
62    properties:
63      options: "${toAccount.type.enumValues}"
64      prompt: "To which account do you want to make a payment?"
65      variable: "toAccount"
66    transitions: {}
67  resolvePaymentAmount:
68    component: "System.SetVariable"
69    properties:
70      variable: "paymentAmount"
71      value: "${iResult.value.entityMatches['CURRENCY'][0]}"
72    transitions: {}
73  askPaymentAmount:
74    component: "System.Text"
75    properties:
76      prompt: "What's the payment amount?"
77      variable: "paymentAmount"
78    transitions: {}
79  unresolved:
80    component: "System.Output"
81    properties:
82      text: "Unable to resolve intent!"
```

6. The final state uses a `System.Output` component and the three variables (`paymentAmount`, `toAccount` and `accountType`) to display a message back to the end user.

   To add this state, copy the code from `doPayment.txt` file (located in the `labfiles/code` directory) and then paste it right above the `unresolved` state that's at the bottom of the flow.

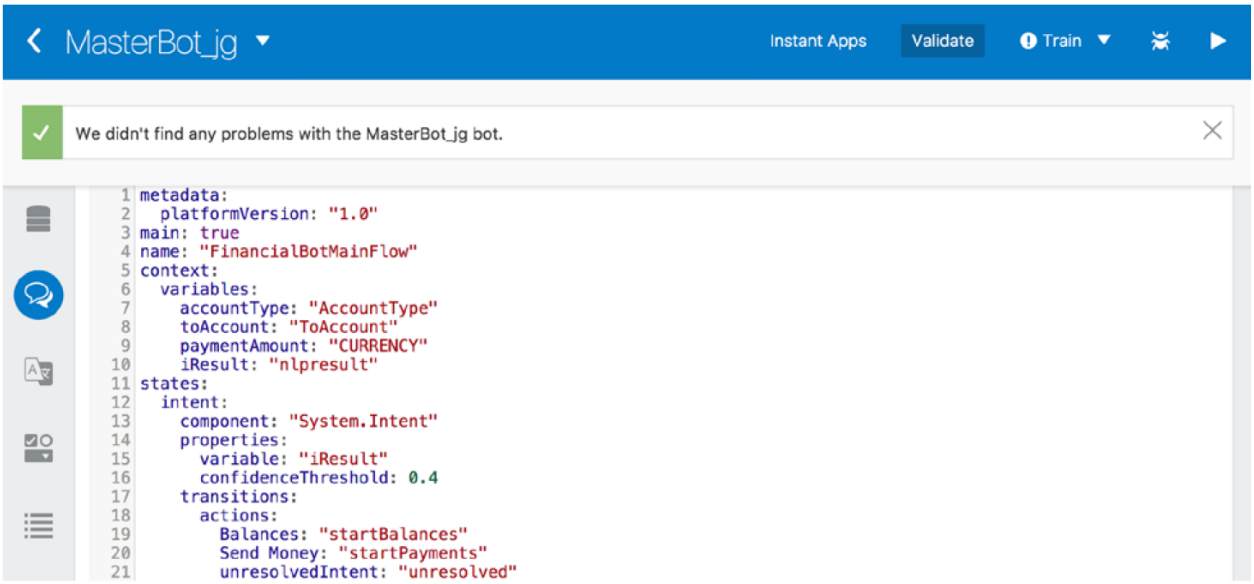
Oracle MOOC: Oracle Intelligent Bots

```
  askPaymentAmount:
    component: "System.Text"
    properties:
      prompt: "What's the payment amount?"
      variable: "paymentAmount"
    transitions: {}
  doPayment:
    component: "System.Output"
    properties:
      text: "${paymentAmount.value.totalCurrency} paid from ${accountType.value} to ${toAccount.value}"
    transitions:
      return: "doPayment"
  unresolved:
    component: "System.Output"
    properties:
      text: "Unable to resolve intent!"
    transitions:
      return: "unresolved"
```

7. Before we test the flow, let's confirm the code is valid. In the upper-right side of the page, click the **Validate** button.

   When validation is complete, you should see a confirmation message that no problems were found. Keep in mind that in many cases errors occur because of indentation.

   Tip: If you run into errors and can't figure out where the problem lies, you may copy and replace the entire BotML definition from the `Lab3_YAML_sol.txt` file, which is located in the `labfiles/code` directory.



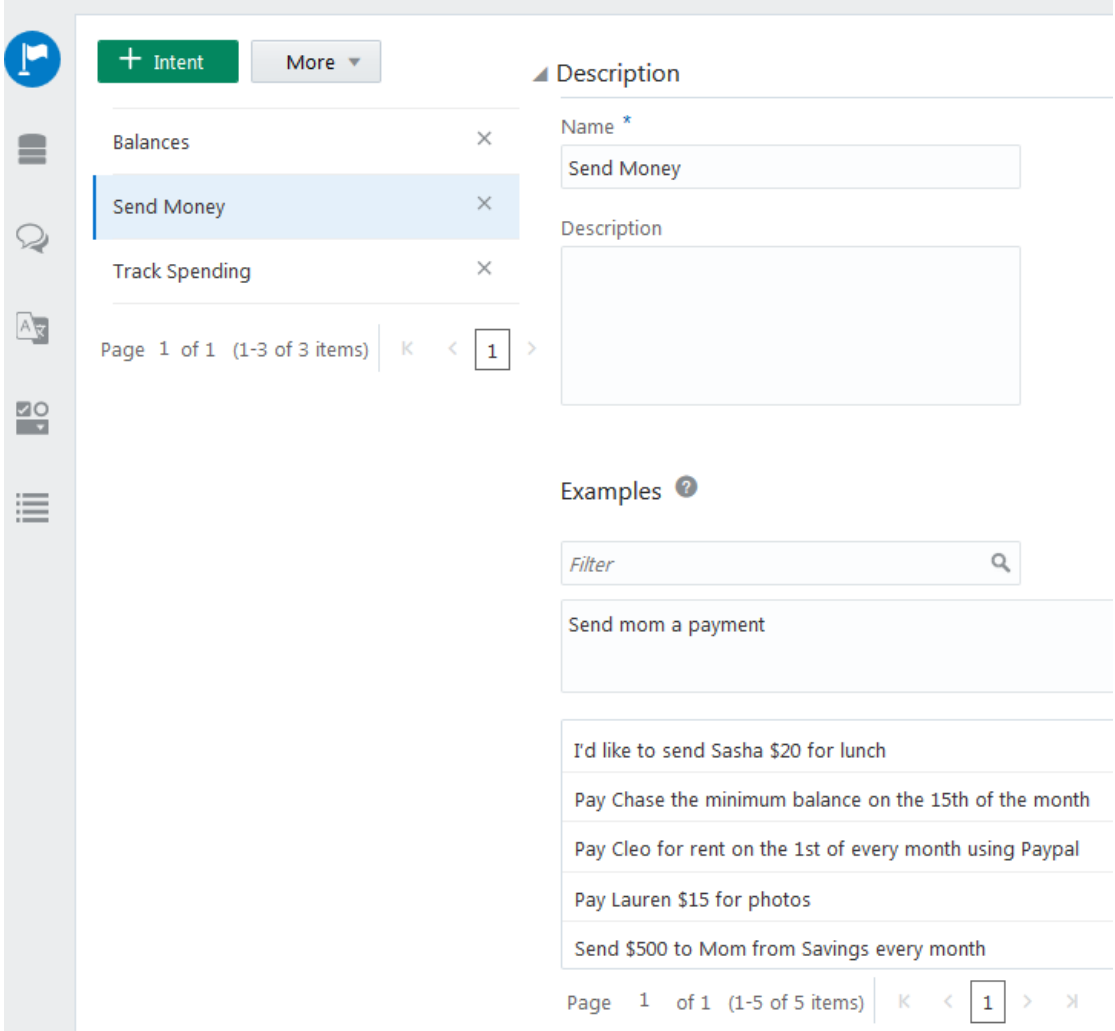
8. Now let's test the flow of the intent. In the upper-right of the page, click the **Play** button to open the Tester (if it's not already open, that is). Be sure to click **Bot**.

9. In the message area, enter *Send a payment* and then click **Send**.

Oracle MOOC: Oracle Intelligent Bots

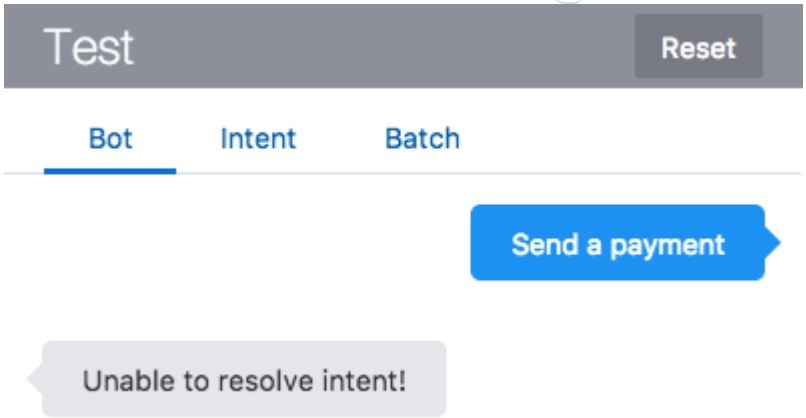
Oracle MOOC: Oracle Intelligent Bots

10. If you receive the *Unable to resolve intent!* error, you will need to add some utterances to an intent like you did in the previous lab. Here, add one or more utterances like “Send mom a payment” to the Send Money intent. When you’re done, click **Train**.



Remember to click **Reset** before you enter the *Send a payment* message again.

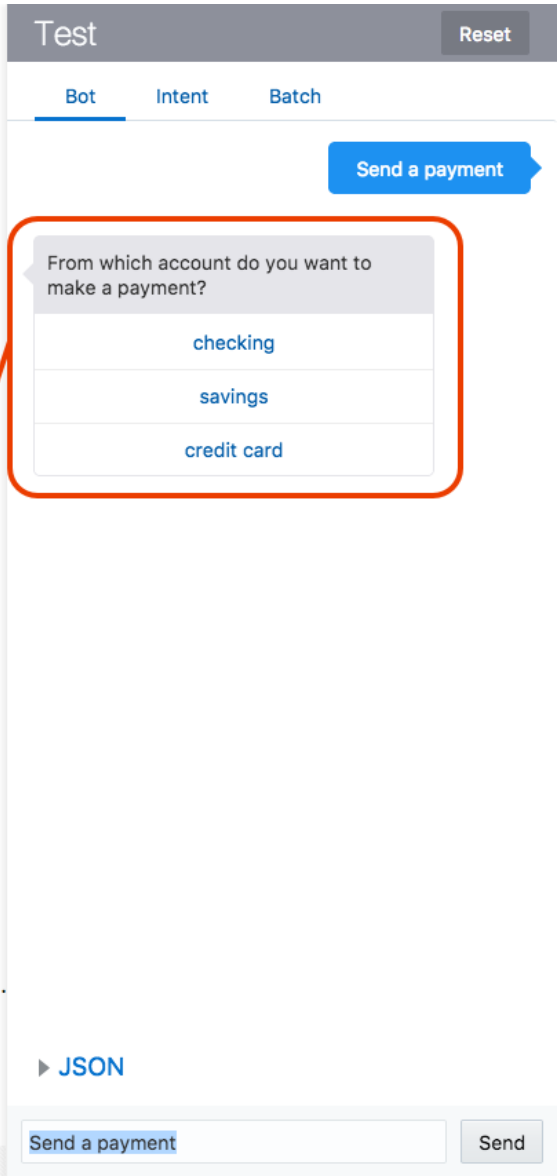
Oracle MOOC: Oracle Intelligent Bots



11. When prompted for an account, either type in the account that you're sending the money from, or select it from the list. The list is presented by the `askFromAccount` state.

Oracle MOOC: Oracle Intelligent Bots

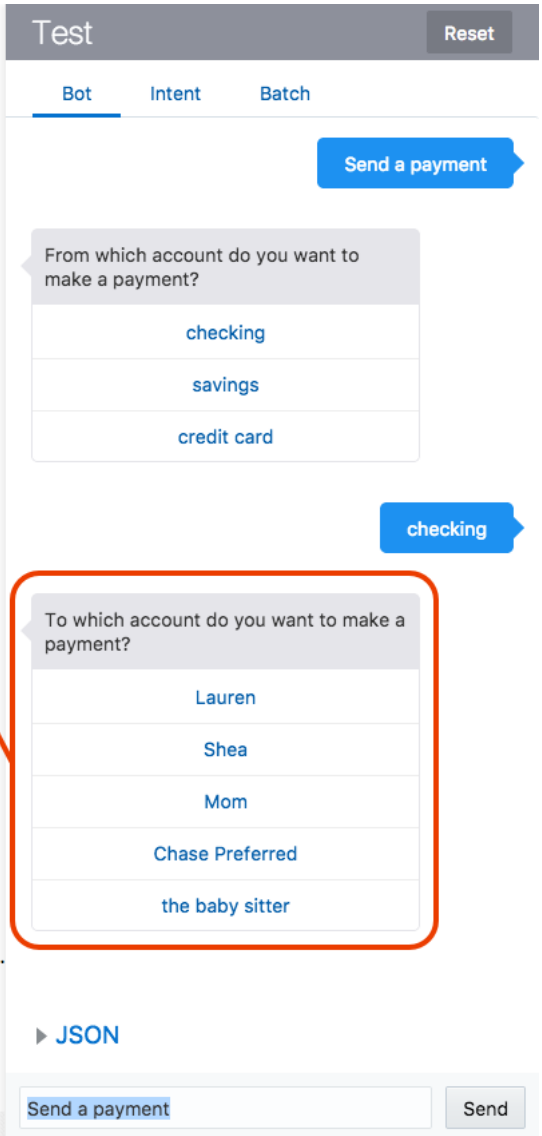
```
  askBalancesAccountType:
    component: "System.List"
    properties:
      options: "${accountType.type.enumValues}"
      prompt: "For which account do you want your balance?"
      variable: "accountType"
    transitions: {}
  printBalance:
    component: "System.Output"
    properties:
      text: "Balance for ${accountType.value} is $500"
    transitions:
      return: "printBalance"
  startPayments:
    component: "System.SetVariable"
    properties:
      variable: "accountType"
      value: "${iResult.value.entityMatches['AccountType'][0]}"
    transitions: {}
  resolveToAccount:
    component: "System.SetVariable"
    properties:
      variable: "toAccount"
      value: "${iResult.value.entityMatches['ToAccount'][0]}"
    transitions: {}
  askFromAccountType:
    component: "System.List"
    properties:
      options: "${accountType.type.enumValues}"
      prompt: "From which account do you want to make a payment?"
      variable: "accountType"
    transitions: {}
  askToAccount:
    component: "System.List"
    properties:
      options: "${toAccount.type.enumValues}"
      prompt: "To which account do you want to make a payment?"
      variable: "toAccount"
    transitions: {}
  resolvePaymentAmount:
    component: "System.SetVariable"
    properties:
      variable: "paymentAmount"
      value: "${iResult.value.entityMatches['CURRENCY'][0]}"
    transitions: {}
  askPaymentAmount:
    component: "System.Text"
    properties:
      prompt: "What's the payment amount?"
      variable: "paymentAmount"
    transitions: {}
  doPayment:
    component: "System.Output"
    properties:
      text: "${paymentAmount.value.totalCurrency} paid from ${accountType.
    transitions:
      return: "doPayment"
  unresolved:
    component: "System.Output"
    properties:
      text: "Unable to resolve intent!"
    transitions:
      return: "unresolved"
```



12. Next, you're prompted for a person to send the money to. Select a person to receive the money. The list is presented by the `askToAccount` state.

Oracle MOOC: Oracle Intelligent Bots

```
  askBalancesAccountType:
    component: "System.List"
    properties:
      options: "${accountType.type.enumValues}"
      prompt: "For which account do you want your balance?"
      variable: "accountType"
    transitions: {}
  printBalance:
    component: "System.Output"
    properties:
      text: "Balance for ${accountType.value} is $500"
    transitions:
      return: "printBalance"
  startPayments:
    component: "System.SetVariable"
    properties:
      variable: "accountType"
      value: "${iResult.value.entityMatches['AccountType'][0]}"
    transitions: {}
  resolveToAccount:
    component: "System.SetVariable"
    properties:
      variable: "toAccount"
      value: "${iResult.value.entityMatches['ToAccount'][0]}"
    transitions: {}
  askFromAccountType:
    component: "System.List"
    properties:
      options: "${accountType.type.enumValues}"
      prompt: "From which account do you want to make a payment?"
      variable: "accountType"
    transitions: {}
  askToAccount:
    component: "System.List"
    properties:
      options: "${toAccount.type.enumValues}"
      prompt: "To which account do you want to make a payment?"
      variable: "toAccount"
    transitions: {}
  resolvePaymentAmount:
    component: "System.SetVariable"
    properties:
      variable: "paymentAmount"
      value: "${iResult.value.entityMatches['CURRENCY'][0]}"
    transitions: {}
  askPaymentAmount:
    component: "System.Text"
    properties:
      prompt: "What's the payment amount?"
      variable: "paymentAmount"
    transitions: {}
  doPayment:
    component: "System.Output"
    properties:
      text: "${paymentAmount.value.totalCurrency} paid from ${accountType.
    transitions:
      return: "doPayment"
  unresolved:
    component: "System.Output"
    properties:
      text: "Unable to resolve intent!"
    transitions:
      return: "unresolved"
```

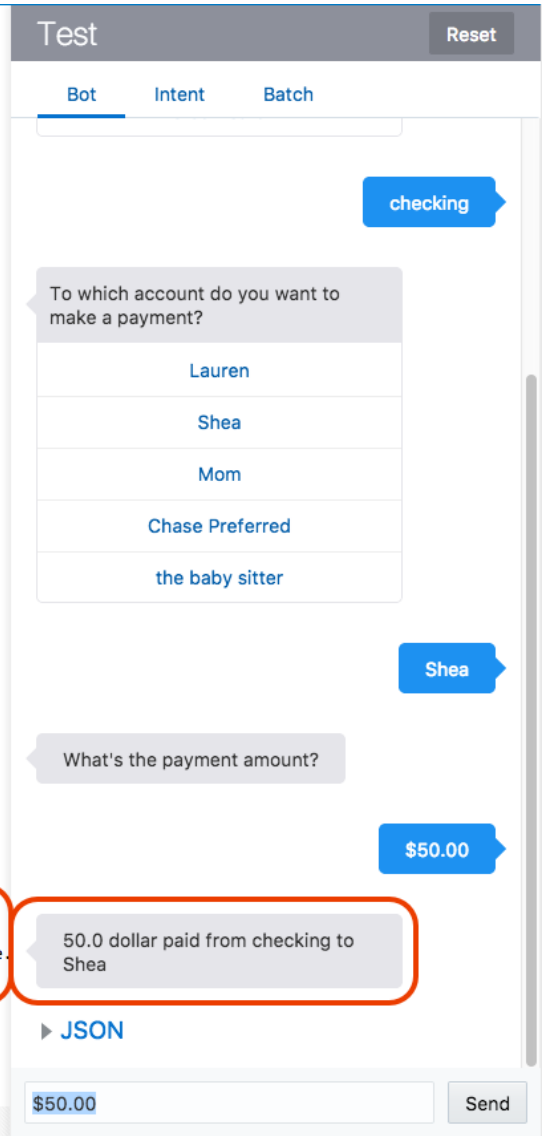


13. Finally, you are prompted for an amount to send. Enter $50.00 and then click **Send**.

A message displays, showing the amount to send, who to send it to, and the account that should send the amount. This is all accomplished by the `doPayment` state.

Oracle MOOC: Oracle Intelligent Bots

```
  transitions: {}
askBalancesAccountType:
  component: "System.List"
  properties:
    options: "${accountType.type.enumValues}"
    prompt: "For which account do you want your balance?"
    variable: "accountType"
  transitions: {}
printBalance:
  component: "System.Output"
  properties:
    text: "Balance for ${accountType.value} is $500"
  transitions:
    return: "printBalance"
startPayments:
  component: "System.SetVariable"
  properties:
    variable: "accountType"
    value: "${iResult.value.entityMatches['AccountType'][0]}"
  transitions: {}
resolveToAccount:
  component: "System.SetVariable"
  properties:
    variable: "toAccount"
    value: "${iResult.value.entityMatches['ToAccount'][0]}"
  transitions: {}
askFromAccountType:
  component: "System.List"
  properties:
    options: "${accountType.type.enumValues}"
    prompt: "From which account do you want to make a payment?"
    variable: "accountType"
  transitions: {}
askToAccount:
  component: "System.List"
  properties:
    options: "${toAccount.type.enumValues}"
    prompt: "To which account do you want to make a payment?"
    variable: "toAccount"
  transitions: {}
resolvePaymentAmount:
  component: "System.SetVariable"
  properties:
    variable: "paymentAmount"
    value: "${iResult.value.entityMatches['CURRENCY'][0]}"
  transitions: {}
askPaymentAmount:
  component: "System.Text"
  properties:
    prompt: "What's the payment amount?"
    variable: "paymentAmount"
  transitions: {}
doPayment:
  component: "System.Output"
  properties:
    text: "${paymentAmount.value.totalCurrency} paid from ${accountType.
  transitions:
    return: "doPayment"
unresolved:
  component: "System.Output"
  properties:
    text: "Unable to resolve intent!"
  transitions:
    return: "unresolved"
```



Next, follow the same process to add the code to support the Track Spending intent.

Step 3: Add the Code to Support the Track Spending Intent

In this section, you'll add the variables and define a start state for the Track Spending intent. Then you'll stub out the Track Spending start state. You'll populate this state with detailed code in a later lab.

If you remember, we added a few entity associations to the Track Spending intent: a date, a date specifier, and track spending category. We need a variable to keep track of the spending categories.

Oracle MOOC: Oracle Intelligent Bots

Important: To avoid any odd characters copied into the flow, type in the values for the next couple of steps rather than using copy and paste.

1. In the `variables` section, add a new variable:

   `spendingCategory: "TrackSpendingCategory"`

```
metadata:
  platformVersion: "1.0"
main: true
name: "FinancialBotMainFlow"
context:
  variables:
    accountType: "AccountType"
    toAccount: "ToAccount"
    spendingCategory: "TrackSpendingCategory"
    paymentAmount: "CURRENCY"
    iResult: "nlpresult"
states:
```

2. Next, we need to define an entry point for the `TrackSpending` intent.

   Locate the `intent` state. In its `transitions` section, add the entry point to the Track Spending intent just below the `Send Money` action:

   `Track Spending: "startTrackSpending"`

```
    iResult: "nlpresult"
states:
  intent:
    component: "System.Intent"
    properties:
      variable: "iResult"
      confidenceThreshold: 0.4
    transitions:
      actions:
        Balances: "startBalances"
        Send Money: "startPayments"
        Track Spending: "startTrackSpending"
        unresolvedIntent: "unresolved"
  startBalances:
    component: "System.SetVariable"
```

3. Now that we have a variable and a starting point for the intent, we need to include the code for the state along with the other states that we've already defined. This code includes a `System.Output` component a place to add details to track the spending, and a return to populate the `trackSpending` variable.

Oracle MOOC: Oracle Intelligent Bots

4. Add the code below to define the details for the `startTrackSpending` state. Place it between the `doPayment` and the `unresolved` states. You can type in this code, or copy and paste from the `startTrackSpending.txt` file, located in the `labfiles/code` file. This is just a placeholder. We'll complete it in the next lab.

```
  doPayment:
    component: "System.Output"
    properties:
      text: "${paymentAmount.value.totalCurrency} paid from ${accountType.value} to ${toAccount.value}"
    transitions:
      return: "doPayment"
  startTrackSpending:
    component: "System.Output"
    properties:
      text: "Resolve the TrackSpending Intent here."
    transitions:
      return: "trackSpending"
  unresolved:
    component: "System.Output"
    properties:
      text: "Unable to resolve intent!"
    transitions:
      return: "unresolved"
```

Good for you! You have now completed this lab. In the next lab, you'll learn how to add custom components to your chatbot.