2.4 Redux Core Concept

Redux itself is very simple. The state of the app we created in the last article can be represented as a generic object like this:

```
{
  carstats: [
    {
      miles: 246,
      model: '60'
    },
    {
      miles: 250,
      model: '60D'
    },
    {
      miles: 297,
      model: '75'
    },
    {
      miles: 306,
      model: '75D'
    },
    {
      miles: 336,
      model: '90D'
    },
    {
      miles: 376,
      model: 'P100D'
    }
  ],
  config: {
    speed: 55,
    temperature: 20,
    climate: true,
    wheels: 19
  }
}
```

This object is the same as the model without setters.

To change this state in Redux, you must dispatch an `action`.

Actions are plain objects describing **what happened** in the app, and serve as the sole way to describe an **intention to mutate the data**. It's one of the **fundamental design choices** of Redux.

Here are some examples to be implemented in our app soon.

```
{
  type: 'SPEED_UP',
  value,
  step: counterDefaultVal.speed.step,
  maxValue: counterDefaultVal.speed.max
}

{
  type: 'CHANGE_CLIMATE'
}

{
  type: 'CHANGE_WHEEL',
  value
}
```

Forcing all of these state changes into action will give us a clear understanding of what's going on in your app. When something happens, we can see why it happened.

Now we need a function called `reducer` to bind these states and actions together. Reducer is nothing more than a function that takes a state and an action as arguments and returns a **new state**.

In a word:

> ***(state, action)*** => ***state***

Actions only describe that something happened and don't specify **how the application's state changes in response**. This is the job of reducers.

Here is one example of a reducer to implement in our app:

```
function appReducer(state = initialState, action) {
  switch (action.type) {
    case 'CHANGE_WHEEL': {
      console.log('CHANGE_WHEEL');
      const newState = {
        ...state,
        config: {
          climate: state.config.climate,
          speed: state.config.speed,
          temperature: state.config.temperature,
          wheels: action.value
        }
      };
      return updateStats(state, newState);
    }
    default:
      return state
  }
}
```