# **授業のポイント**

1. Stateパターンについて学習します。

# **学習項目**

・状態遷移図と状態遷移表

・Stateパターン

# **ガチャガチャマシーンの不具合**

# **１．ガチャガチャマシーンのリファクタリング**

ガチャガチャマシーン（GachaMachineクラス）に「売り切れ」を追加します。ただし業者が作成したガチャガチャマシーン（J2Kad24B作成前のGachaMachineクラス）に「売り切れ」を追加するには多くのメソッドを修正する必要があります。よってJ2Kad24DからJ2Kad24Bにかけて、ガチャガチャマシーンのリファクタリングを行い、J2Kad24Aで「売り切れ」を追加します。

# **２．状態遷移図（課題プリント参照）**

ガチャガチャマシーンの状態（「コイン：なし」「コイン：あり」＋「売り切れ」）と、どういう操作で状態が遷移するのかを矢印で示した図を状態遷移図といいます。

# **J2Kad24D「状態クラス」**

# **１．Stateパターン**

Stateパターンとは、具体的な「もの」ではなく「状態」をクラスとして表現するパターンです。同じ操作でも状態によって振る舞いが変化する場合に有効です。状態を個々のクラスで管理するため、コードがわかりやすくなります。今回はこのStateパターンを適用します。

# **２．状態クラス**

ガチャガチャマシーンの状態と操作は以下のようになります。

・状態 …コインなし（NoCoin）、コインあり（HasCoin）、売り切れ（SoldOut）※SoldOutはJ2Kad24Aで追加

・操作 …コインを入れる（inserCoin）、ハンドルを回す（turnHandle）、返却ボタンを押す（ejectCoin）

ここではNoCoinとHasCoinを作成します（SoldOutはJ2Kad24Aで追加）。どちらの状態もできる操作は同じなのでGachaStateインターフェイスを作成して実装させます（ついでに状態表示（showState）も追加）。あとはNoCoinとHasCoinの個々のメソッドを作成していくだけです。

# **J2Kad24C「状態遷移」**

* **J2Kad24DのGachaStateインターフェイス、NoCoinクラス、HasCoinクラスをファイル「GachaMachine.java」にコピーして作成させてください。ただしIntelliJの場合、普通にコピペするとパッケージ名まで補完します。よって貼り付ける場所で右クリックして[特殊なコピー/貼り付け]→[プレーンテキストとして貼り付け]で貼り付けさせてください。**

# **１．状態遷移表**

状態遷移図を表形式にしたものが状態遷移表です。ここでは状態遷移表にもとづいて、状態遷移処理を追加します。状態遷移表のすべての項目について動作をチェックすれば仕様通り完成したことになります。なお、課題で指定された順番でのメソッド呼び出しではNoCoin.turnHandleとHasCoin.ejectCoinのチェックが抜けていますが、J2Kad24Bですべてのチェックを行います。

# **J2Kad24B「リファクタリング！」**

# **１．状態クラスへの委譲**

J2Kad24Cまで作成したらStateパターンはほぼ完成です。GachaMachineクラスのshowStateメソッド、insertCoinメソッド、turnHandleメソッド、ejectCoinメソッドのコードを「すべて削除」し、状態クラスへ処理を委譲するだけです。もちろん委譲を使わずに作成する方法もあります。

# **２．Stateパターンの委譲とStrategyパターンの委譲**

今回のStateパターン（解答例①の方）も前々回のStrategyパターンも委譲を使っています。プログラムコードにするとどちらも同じようなコード（他のクラスへ丸投げ）ですが、以下のように考え方が異なります。

・Stateパターン …「状態」に着目して委譲

・Strategyパターン …「アルゴリズム」に着目して委譲

結果的に同じようなコードになりますが、ここで大切なのは何に着目してクラスにするのかという点です。具体的な「もの」だけでなく「状態」や「アルゴリズム」、さらには「手順」や「命令」などもクラスとして考えることができます。

# **J2Kad24A「ガチャガチャマシーン完成！」**

* **pac24bの「GachaMachine.java」をファイルごと上書きコピー作成させてください（pac24aに最初から入っている「GachMachine.java」はエラーが出ないようにするためのダミーファイルです）。**

# **１．「売り切れ」の追加**

「売り切れ」の追加はいたって簡単です。SoldOutクラスを作成してHasCoinクラスの状態遷移処理を修正するだけです。もはやGachaMachineクラスを修正する必要もありません。

# **２．参考：MECE（ミッシー、「漏れなくダブりなく」）**

MECE（Mutually Exclusive and Collectively Exhaustive）とは「漏れなくダブりなく」という意味です。プログラムではロジックに何らかの「漏れ」があるとバグが発生する可能性があります。また「ダブり」があるといたずらに容量を使ったり場合によっては余分に処理時間がかかったりします。MECEなコードを書くというのは非常に重要で、容量や処理時間に効果があるだけでなく動作テストやデバッグも容易になります。

状態遷移図を描くことによって処理の流れが合っているかどうか確認できます。状態遷移表を作成することによって「漏れなくダブりなく」すべての処理が網羅されることになります。コーディングはこれらひとつひとつを作成していけばよく、動作テストも各項目が正常に動作するかどうかを確認すれば全体を確認できます。プログラム全体の信頼性が向上します。

# **J2Kad24S「ストップウォッチ！（State版）」**

# **１．ストップウォッチの状態遷移表**

前期課題JKad23X「ストップウォッチ！」を、Stateパターンを使って作成します。switch文やif文を使ってがんばって作成したコードが、オブジェクト指向ではこうなるという例です。課題プリントの状態遷移図をもとに状態遷移表を作成し、あとはひとつひとつの項目のコードを作成していくだけです（showStateも忘れずに）。

**ストップウォッチの状態遷移表**

|  |  |  |  |
| --- | --- | --- | --- |
| 状態 | start | clear | noAction |
| Stop | 「計測を始めます！」→Runへ | 「何も起こりません！」 | 「止まっています・・・」 |
| Run | 「一時停止します！」→Pauseへ | 「何も起こりません！」 | 「計測中です・・・」 |
| Pause | 「計測を再開します！」→Runへ | 「タイムをリセットして停止します！」→Stopへ | 「一時停止中です・・・」 |

# **J2Kad24X「世界にはばたくECCフーズ！」※J2Challenge12Sのリマインド、次回解答編**

# **１．Iteratorパターン**

Iteratorパターンとは、コンテナオブジェクトの要素を列挙する手段を独立させることによってコンテナの内部仕様に依存しない反復子（イテレータ）を提供するパターン（Wikipediaより）です。たぶんこの解説では何を言っているのかわからないと思うので、ここではどんな形であれ仕様通りに動作していればOKとします。次回、解答編です。