【補足】
Spring Bootアプリケーションのテスト

Ver3.2.1-2

JUnit

- Javaでテストを記述する際のライブラリ(テスティングフレームワークとも呼ばれる)
- 類似したライブラリも存在するが、JUnitがデファクトスタンダートとなっている

JUnit

- JUnitを使用したテストコードの例

【テスト対象のクラス】

```
public class SampleService {
    public int plus(int a, int b) {
        return a + b;
    }
}
```

【テストコード】

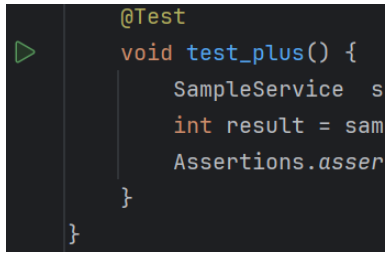
```
class SampleServiceTest {
    @Test   ・・・①
    void test_plus() {
        SampleService  sampleService = new SampleService();
        int result = sampleService.plus(3, 4);
        Assertions.assertThat(result).isEqualTo(7);   ・・・②
    }
}
```

①	テストを実施するメソッドに付けるアノテーション。メソッド内でテスト対象の処理を呼び出す
②	処理の結果を確認

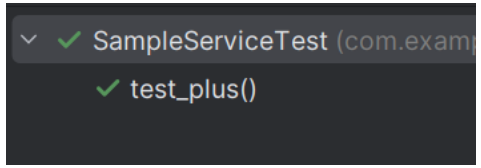
JUnit

- テスト実行の流れ

実行ボタンを表示したIDEの画面



結果を表示したIDEの画面



JUnit

- 複数のテストメソッド

【テスト対象のクラス】

```
public class SampleService {
    public int plus(int a, int b) {
        return a + b;
    }
    public int minus(int a, int b) {
        return a - b;
    }
}
```

【テストコード】

```
class SampleServiceTest {
    SampleService sampleService;  ・・・①
    @BeforeEach  ・・・②
    void setUp() {
        sampleService = new SampleService();
    }
    @Test
    void test_plus() {
        int result = sampleService.plus(3, 4);
        Assertions.assertThat(result).isEqualTo(7);
    }
    @Test
    void test_minus() {
        int result = sampleService.minus(7, 4);
        Assertions.assertThat(result).isEqualTo(3);
    }
}
```

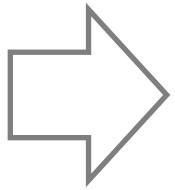
①	テスト対象のオブジェクトを入れるフィールド
②	テストの前処理のメソッドに付けるアノテーション。メソッド内でテスト対象のオブジェクトを用意したりする

JUnit

- @BeforeEachの挙動
  - 各テストメソッドごとに毎回呼び出される

```
class SampleTest {
    @BeforeEach
    void setUp() {
        System.out.println("setUp");
    }
    @Test
    void test1() {
        System.out.println("test1");
    }
    @Test
    void test2() {
        System.out.println("test2");
    }
}
```

実行



コンソール

```
setUp
test1
setUp
test2
```

JUnit

- 実行結果が期待通りかどうかを確認することを「アサーション」と言う
  - AssertJというライブラリを使用したサンプル

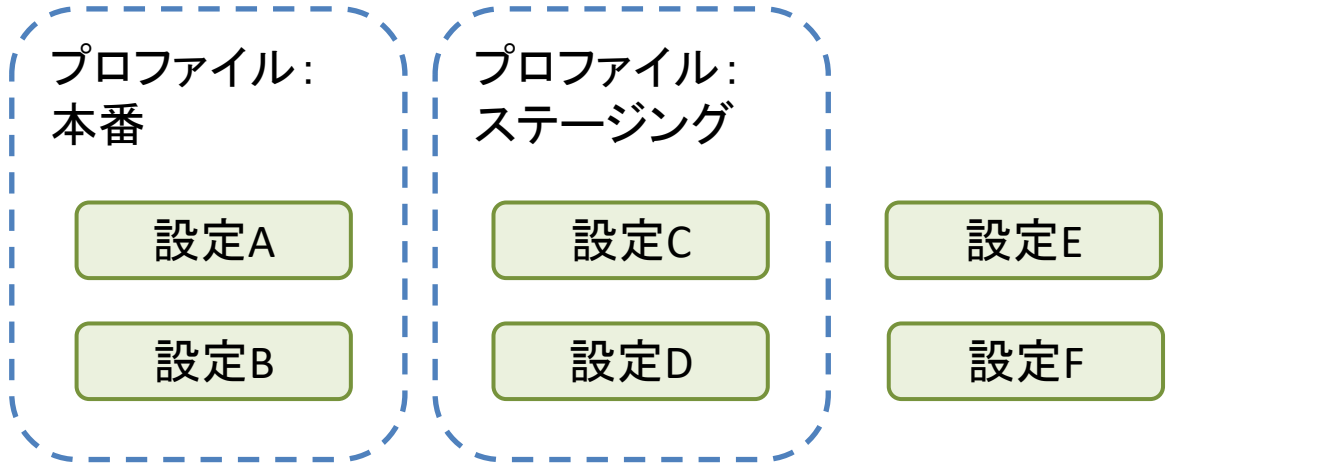
```
@Test
void test_selectOrder() {
    Order order = orderRepository.selectById("o01");
    Assertions.assertThat(order.getCustomerName()).isEqualTo("cname01");
}
```

  - Assertionsクラスのstaticメソッドをstaticインポートするとよい

```
import static org.assertj.core.api.Assertions.*;
...
@Test
void test_selectOrder() {
    Order order = orderRepository.selectById("o01");
    assertThat(order.getCustomerName()).isEqualTo("cname01");
}
```

プロファイル

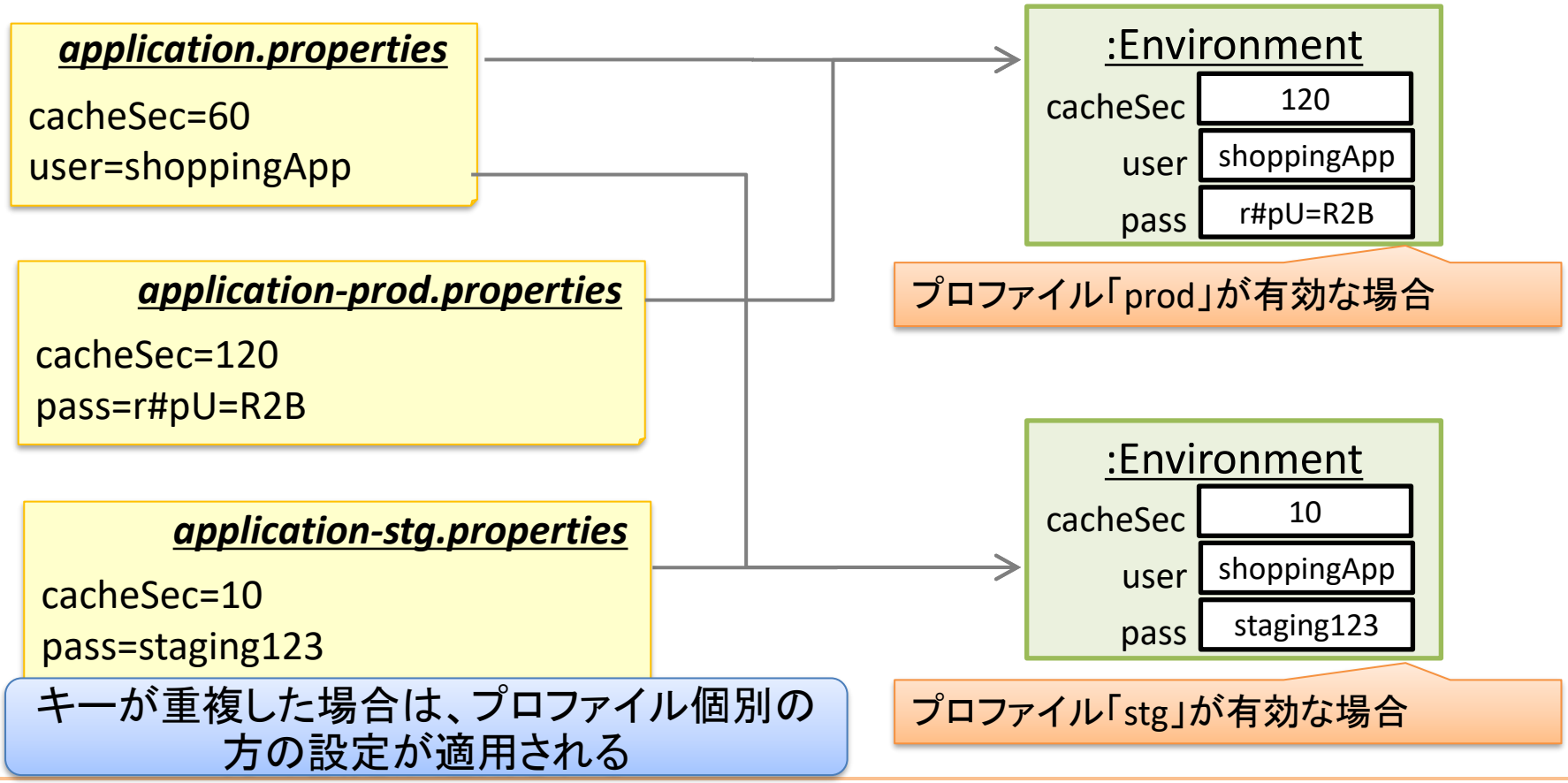
- プロファイルとは
  - コンフィグレーション(DIコンテナが読み込む設定情報)をグルーピングするSpringの仕組み
- アプリケーション起動時にどのプロファイルを有効にするかを指定できる



「本番」を有効にして起動した場合は、設定A、Bと、設定E、Fが有効になる(プロファイルに属していない設定は常に有効)

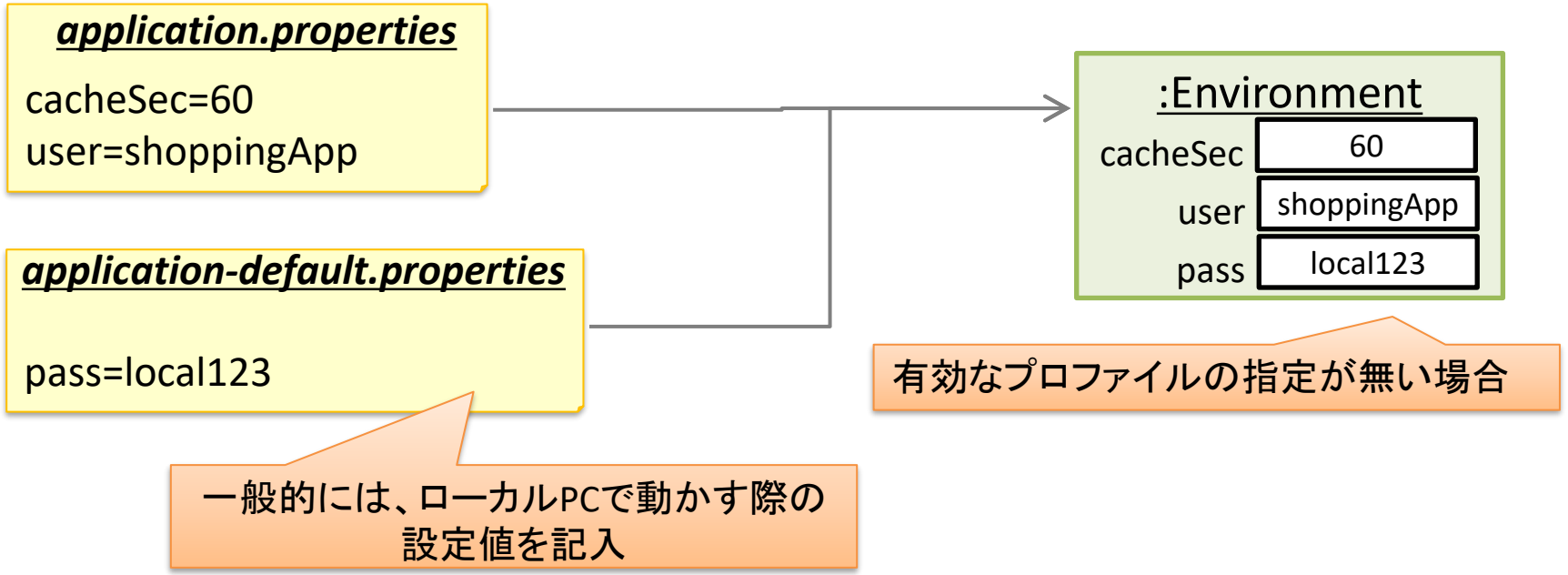
プロファイル

- application-プロファイル名.propertiesを作成すると、有効なプロファイルとして「プロファイル名」が指定されたときに読み込まれる



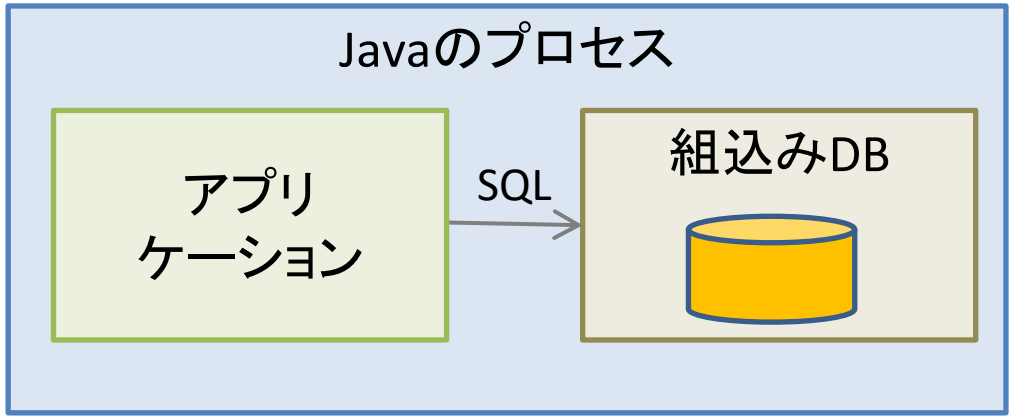
プロファイル

- 有効にするプロファイルを何も指定せずにDIコンテナを生成した場合は、自動的に「default」というプロファイルが有効になる



組み込みデータベース

- アプリケーションと同じJavaプロセスの中で動く、Javaで作られたデータベース
  - ライブラリとして取り込んで使うことができる
- データベースを事前にインストールする手間がない
- データは基本的にメモリ上で保持されるため、プロセスが終了するとデータも消える
  - 開発・テスト用途で使用する
- 代表的な製品として、H2、HSQLDB、Derbyなどがある



JSON

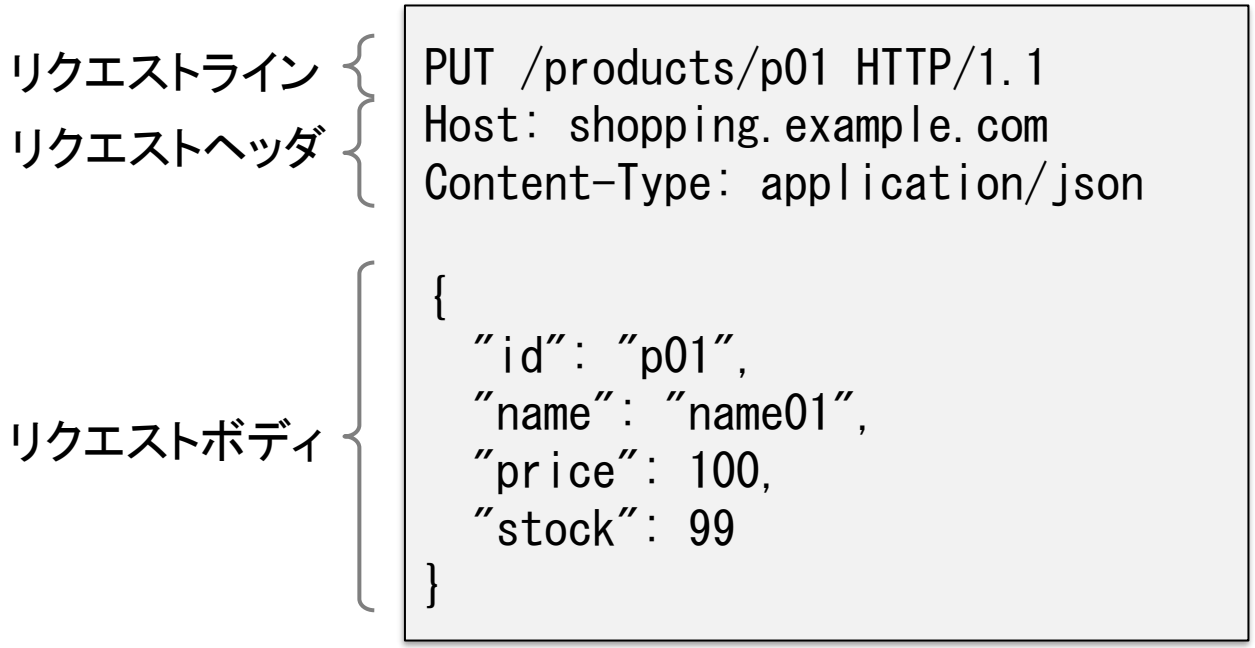
- さまざまな用途で使用される標準化されたテキストデータの形式。代表的な用途として、Webサービスで送受信する際のデータ形式がある
- JSONという単語は、JavaScript Object Notationの略語。元々はJavaScriptのオブジェクトをテキストで表現するための形式だが、シンプルで分かりやすいため、現在はさまざまな用途で使用されている

```
{
  "employeeId":"emp01",
  "employeeName":"東京太郎",
  "age":25,
  "projects":[
    {"projectId":"proj01", "projectName":"Xシステム開発"},
    {"projectId":"proj02", "projectName":"Yシステム開発"}
  ]
}
```

1つ1つのデータを「メンバ」と呼ぶ

HTTPのデータ構造

- リクエストのデータ構造



HTTPのデータ構造

- レスポンスのデータ構造

ステータスライン
レスポンスヘッダ
レスポンスボディ

```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Tue, 03 Jan 2023 04:52:29 GMT

{
  "id": "p01",
  "name": "name01",
  "price": 100,
  "stock": 10
}
```

JSONPath

- JSONデータの中から特定箇所を抽出する際の標準化された書式

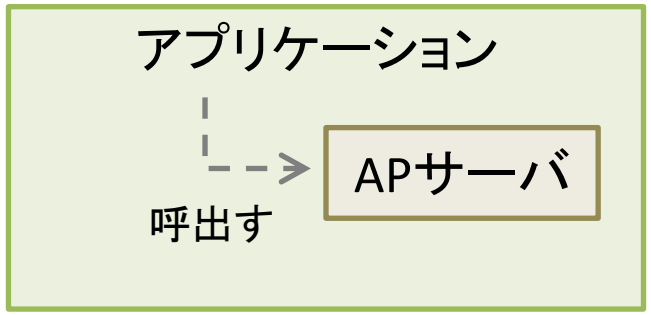
```
{
  "id" : "o01",
  "customerName" : "東京太郎",
  "customerAddress" : "address01",
  "orderItems" : [ {
    "id" : "i01",
    "priceAtOrder" : 1000,
    "product" : {
      "id" : "p01",
      "name" : "name01"
    }
   }, {

    "id" : "i02",
    "priceAtOrder" : 2000,
    "product" : {
      "id" : "p02",
      "name" : "name02"
    }
  } ]
}
```

サンプル	取得される値
$.customerName	東京太郎
$.orderItems[0].priceAtOrder	1000
$.orderItems[1].product.name	name02
$.orderItems.length()	2

組み込みAPサーバ

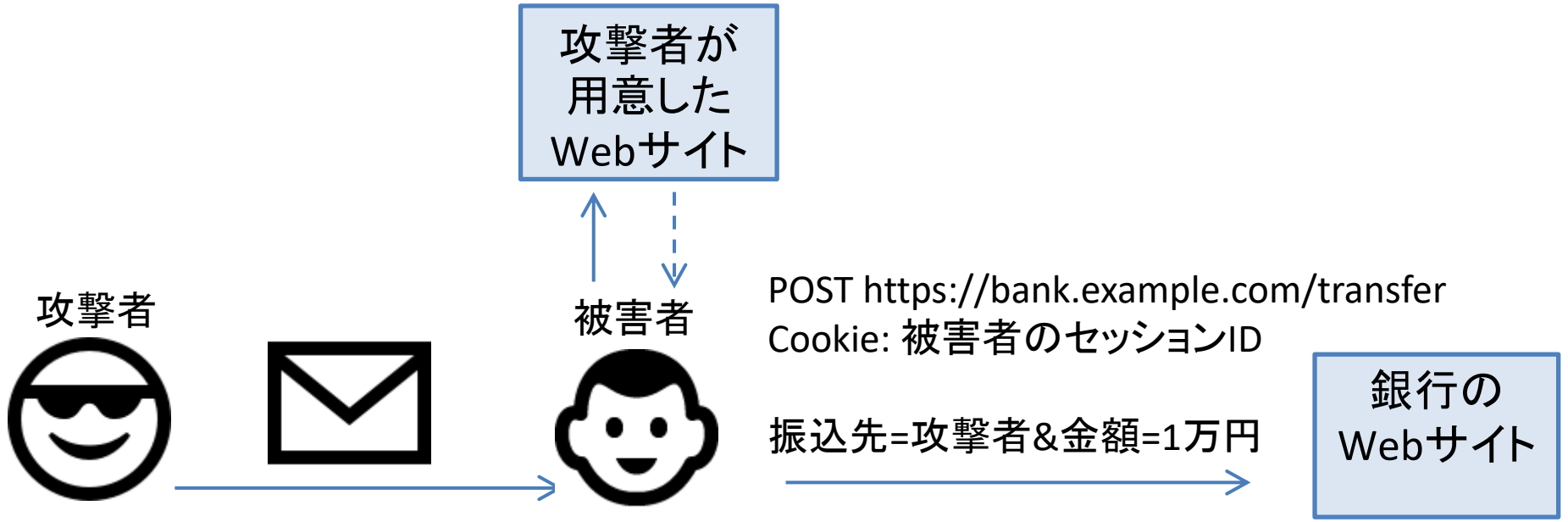
- アプリケーションに組み込まれたAPサーバ
- APサーバはJavaで作られているため、アプリケーションがAPサーバをライブラリとして取り込み、APサーバの起動用のメソッドをアプリケーション側から呼び出して起動する
  - 事前にAPサーバをインストールする必要がない



CSRF

- CSRF(Cross Site Request Forgery)とは？
  - Webサイトに対する代表的な攻撃の1つ
  - 被害者はWebサイト上で意図しない操作をさせられてしまう

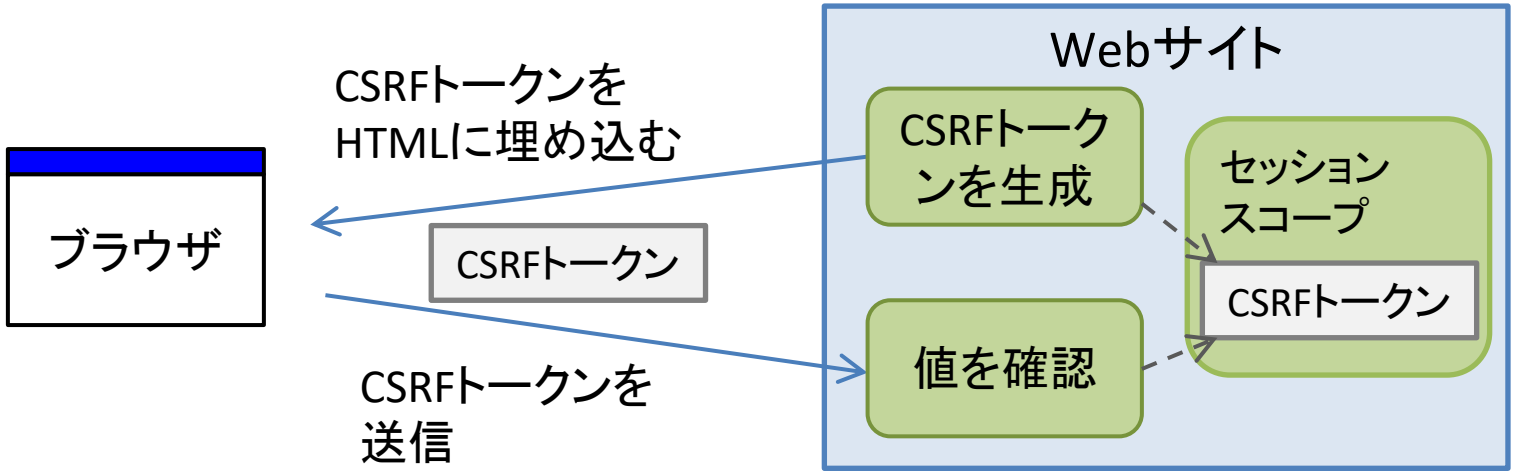
CSRF




```
<h1>100万円ゲット！！</h1>
<form action="https://bank.example.com/transfer" method="post">
  <input type="hidden" name="振込先" value="攻撃者"/>
  <input type="hidden" name="金額" value="1万円"/>
  <input type="submit" value="今すぐゲット"/>
</form>
```

CSRFの一般的な対策

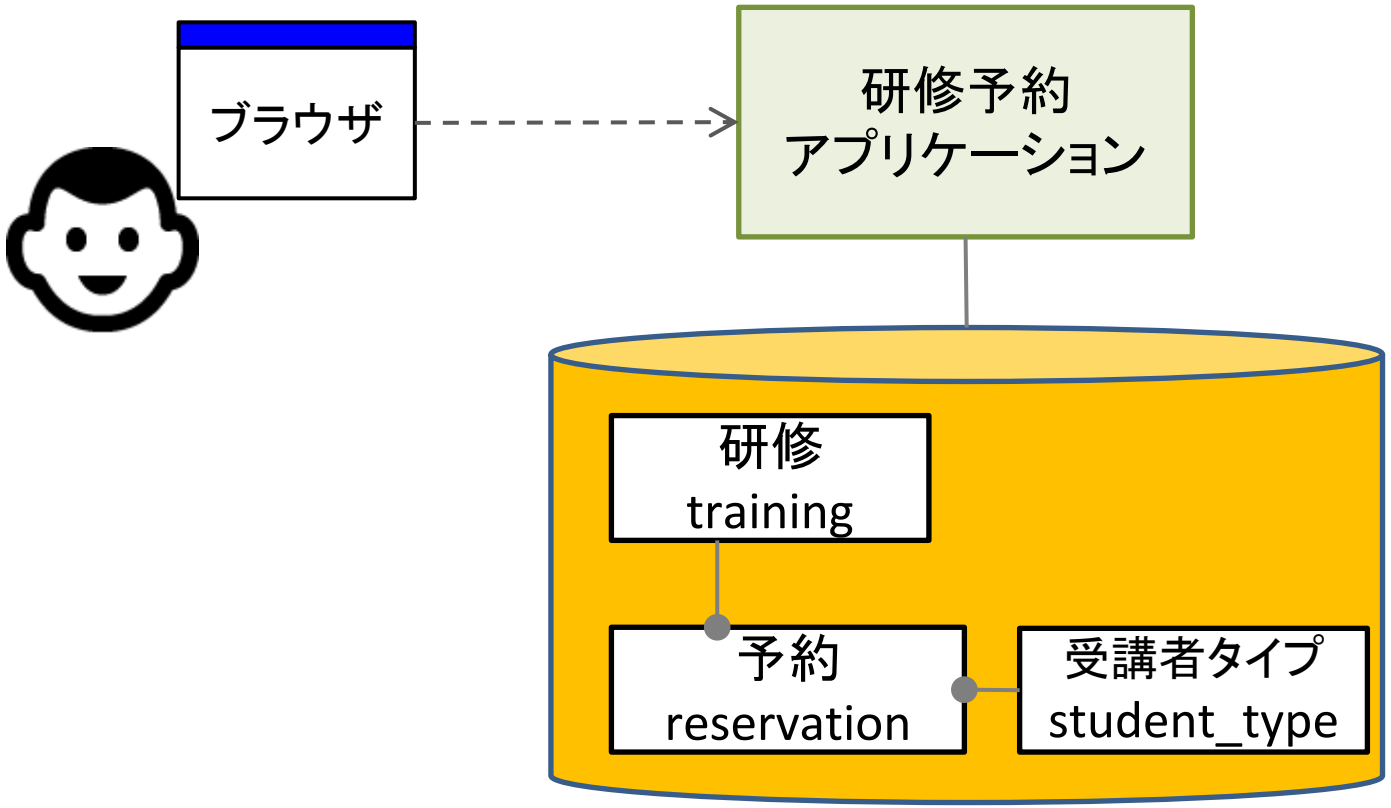
- CSRFトークンと呼ばれる秘密情報をリクエストに含めてもらう



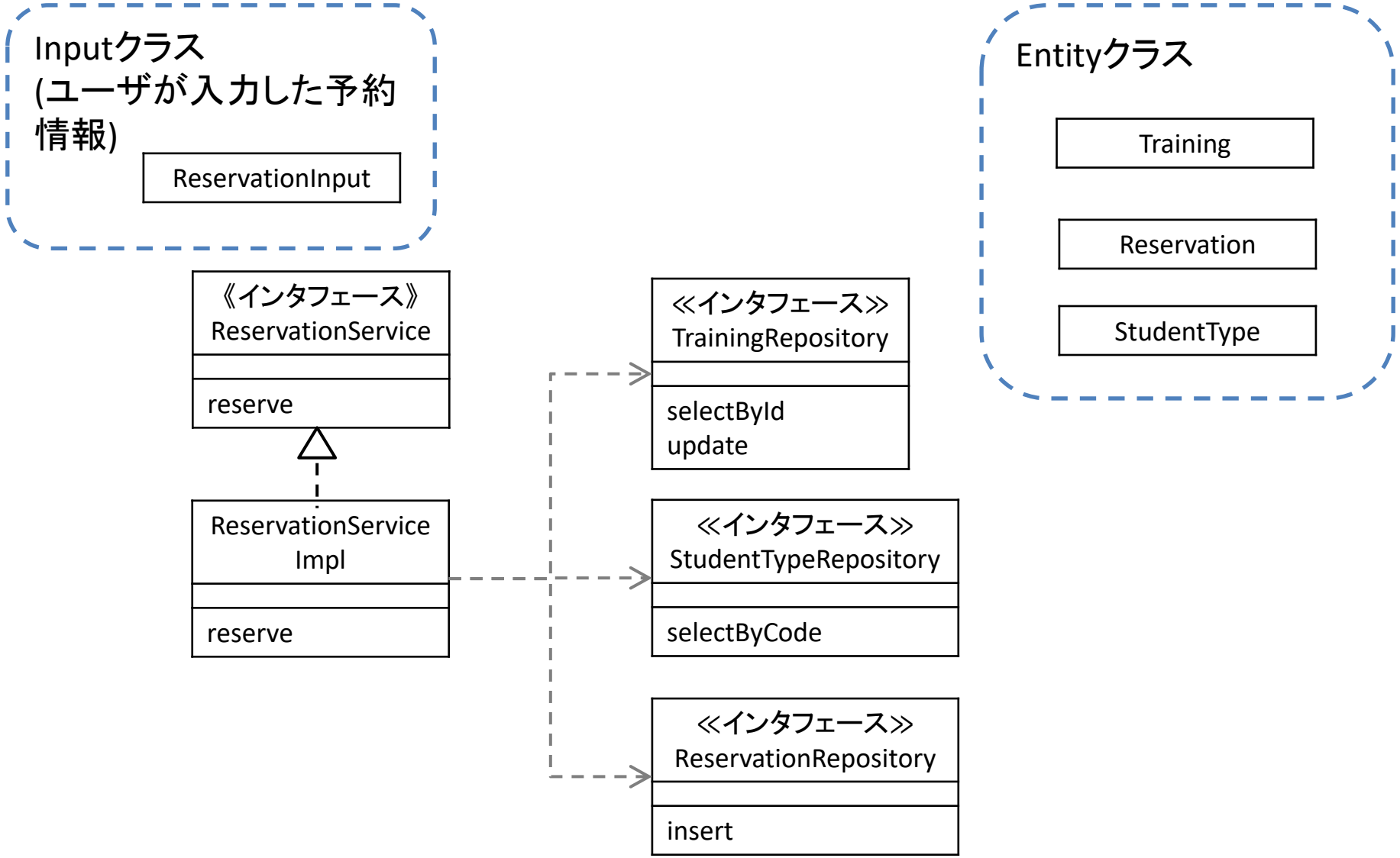
- 攻撃者はCSRFトークンを知ることができないため、悪意のあるリクエストを作成できない

研修予約アプリケーション

- ブラウザを操作して研修を予約するアプリケーション



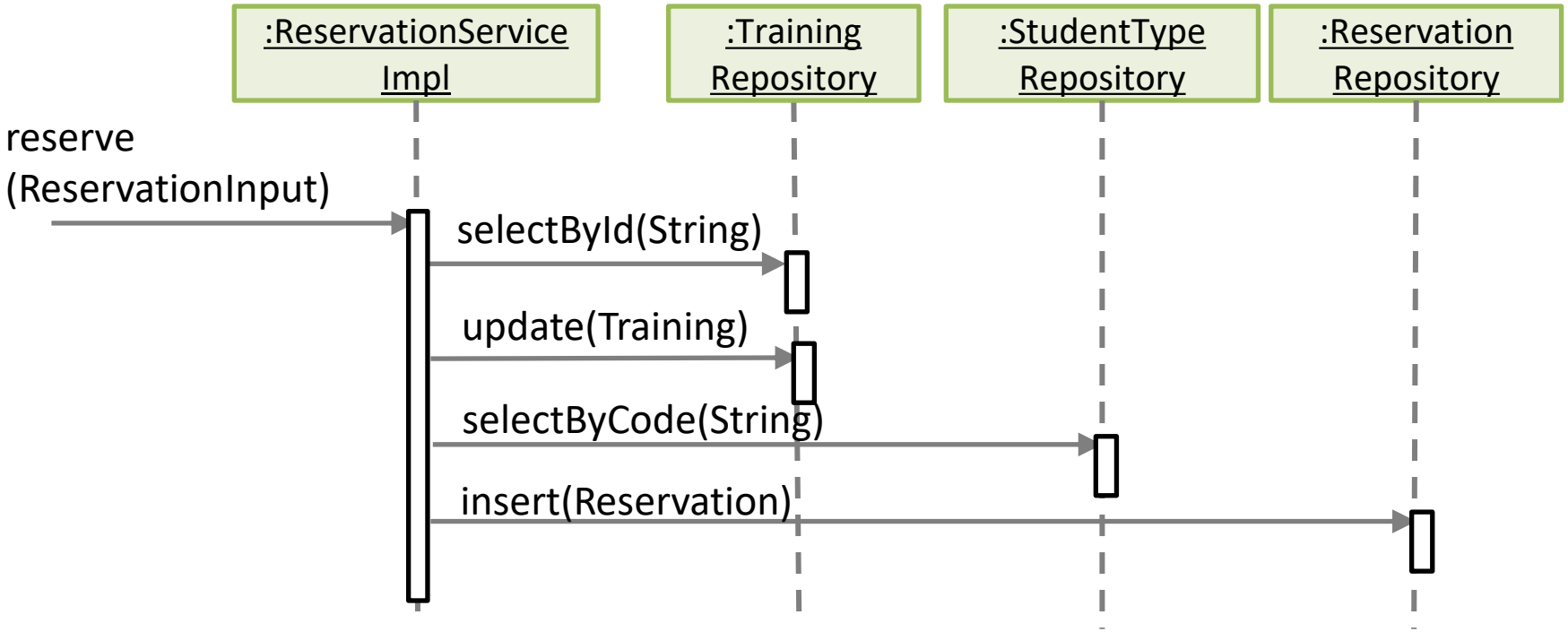
研修予約アプリケーション



研修予約アプリケーション

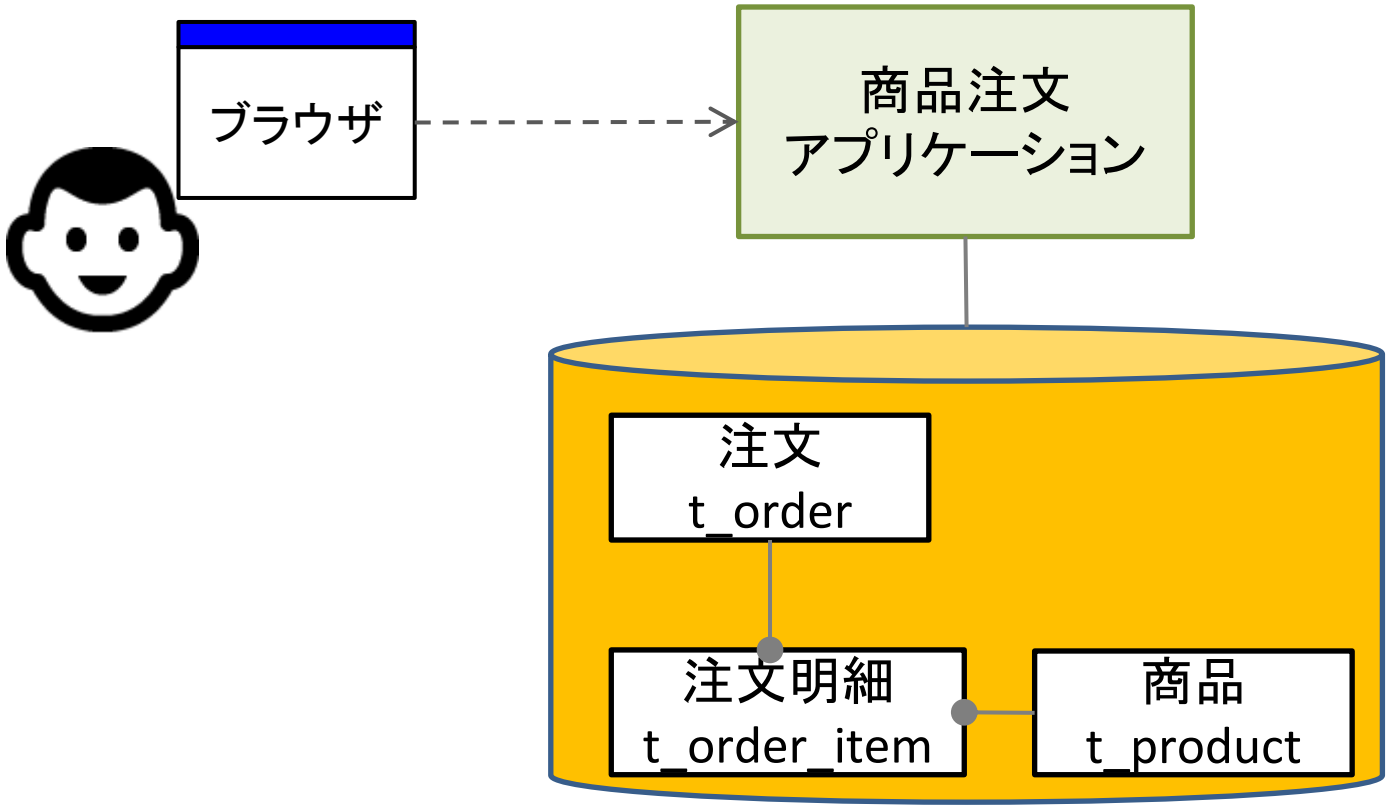
クラス・インタフェース	役割
ReservationInput	ユーザが入力した予約情報(予約する研修の情報や、ユーザの情報など)を保持するInputクラス
ReservationService	研修予約メソッド(reserveメソッド)を定義したServiceインタフェース。ReservationInputを引数で受け取る
ReservationServiceImpl	研修予約メソッドを実装したServiceクラス
TrainingRepository	研修データのデータベースアクセスを行うためのRepositoryインタフェース
ReservationRepository	予約データのデータベースアクセスを行うためのRepositoryインタフェース
StudentTypeRepository	受講者タイプデータのデータベースアクセスを行うためのRepositoryインタフェース
Training	研修データを保持するEntityクラス
Reservation	予約データを保持するEntityクラス
StudentType	受講者タイプデータを保持するEntityクラス

研修予約アプリケーション

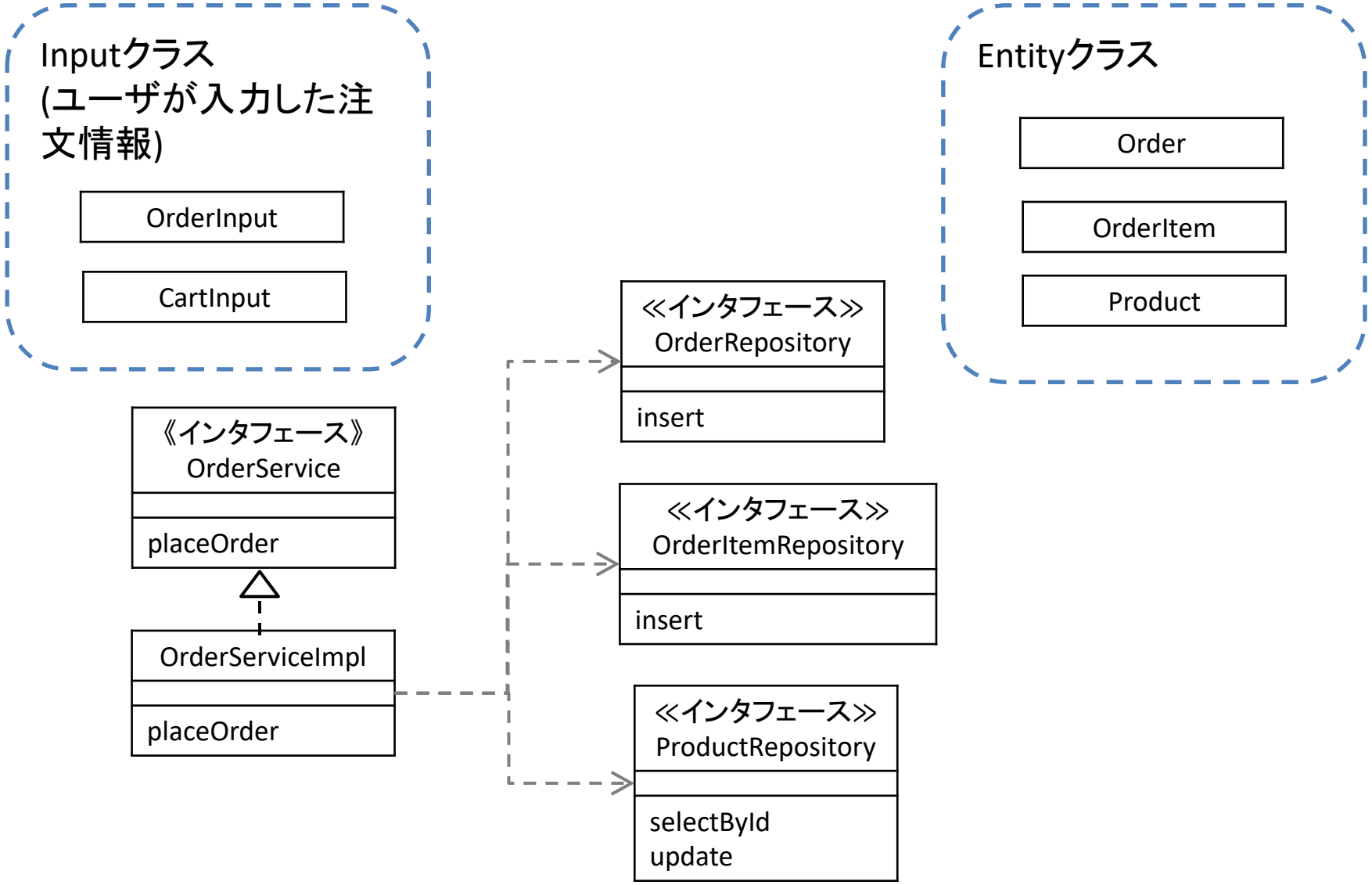


商品注文アプリケーション

- ブラウザを操作して商品を注文するアプリケーション



商品注文アプリケーション



商品注文アプリケーション

クラス・インタフェース	役割
OrderInput	ユーザが入力した注文情報(ユーザの名前や住所など)を保持するInputクラス
CartInput	ユーザが注文する商品情報を保持するInputクラス
OrderService	商品注文メソッド(placeOrderメソッド)を定義したServiceインタフェース。OrderInputとCartInputを引数で受け取る
OrderServiceImpl	商品注文メソッドを実装したServiceクラス
OrderRepository	注文データのデータベースアクセスを行うためのRepositoryインタフェース
OrderItemRepository	注文明細データのデータベースアクセスを行うためのRepositoryインタフェース
ProductRepository	商品データのデータベースアクセスを行うためのRepositoryインタフェース
Order	注文データを保持するEntityクラス
OrderItem	注文明細データを保持するEntityクラス
Product	商品データを保持するEntityクラス

商品注文アプリケーション