第三回　演習問題（抽象クラス，インタフェース）

諸注意

- 「Arms.java」，「Item1.java」，「Usable.java」，「Item2.java」，「Sword.java」，「Cane.java」，「Harb.java」，「JokeTester.java」を Web から提出する（８ファイルを提出する場合，８回に分けて提出処理を行う）．
- コピペ発覚時は見せた側も見せてもらった側も両方０点とする．
- 必ずコンパイルエラーのない状態で提出すること（自動採点したいのでコンパイルエラーがあると，全て０点になってしまう）．
- 課題の途中で提出することになった場合，コンパイルエラーさえ出なければ，課題の途中の状態で提出してくれて構わない．一部のメソッドだけが実現できていない場合，コンパイルエラー出ないならばそのままの状態で提出してくれてよい．
- 主にコンソール出力で評価しているため，デバッグに用いたようなコンソール出力が残っていないように気をつけること．基本的にコンソール出力を指定しない限りは，課題内でコンソール出力はないものとする．
- Package は使わないこと（デフォルトパッケージで実装する）．Package で実装すると，自動採点がうまくいきません．

課題 1

1-1	問題設定	ロールプレイングゲームの武器をイメージしてほしい．色々な武器を考えた結果，抽象クラス Arms を作成し，それぞれの武器がこれを継承することで，統一性のある武器クラスを開発できると考えた．以下の条件で Arms を開発せよ．ただしフィールドは全て隠蔽（自身を継承した子クラスからはアクセスできるように）し，メソッドは他のパッケージからでもアクセスできるようにせよ．
	フィールド	name: 武器の名称(文字列型) ap: AttackPoint: 攻撃力（数値型） map: MagicAttackPoint: 魔法攻撃力（数値型）
	コンストラクタ	1．全てのフィールドの初期値を設定する．
	メソッド	int getAp(): 攻撃力を取得するメソッド 　引数　：なし 　戻り値：攻撃力 int getMap(): 魔法攻撃力を取得するメソッド 　引数　：なし 　戻り値：魔法攻撃力 String toString(): 武器の詳細を確認するメソッド 　引数　：なし 　戻り値：「武器名(攻撃力,魔法攻撃力)」 　※括弧（）とカンマ，は半角を用いること． 　※不要なスペース，半角スペースは表示しないこと． void attackEffect(): 攻撃のエフェクトを表示するメソッド 　引数，戻り値：なし 　※抽象メソッドとする．

1-2	問題設定	ロールプレイングゲームの道具を管理する抽象クラスItem1を開発せよ．ただしフィールドは全て隠蔽（自身を継承した子クラスからはアクセスできるように）し，メソッドは他のパッケージからでもアクセスできるようにせよ．
	フィールド	name: 道具の名称(文字列型) description: 道具の説明(文字列型)
	コンストラクタ	1．全てのフィールドの初期値を設定する．
	メソッド	String toString(): 道具の詳細を確認するメソッド 引数　：なし 戻り値：「道具名(道具の説明)」 ※括弧（）は半角を用いること． ※不要なスペース，半角スペースは表示しないこと． void useEffect(): 道具使用のエフェクトを表示するメソッド 引数，戻り値：なし ※抽象メソッドとする．

1-3	問題設定	よくよく考えると，道具として使用できる武器というものが存在することに気づいた．すなわち，Arms を継承してできる武器の中に Item クラス同様に useEffect()が使用できるクラスを開発する可能性がある．これを実現するためのインタフェース Usable を定義せよ． また，Usable を定義したことで，Item1 の一部を Usable で定義できることになる．修正したヴァージョンである Item2 を定義せよ．（Item1 をそのまま修正されると，課題 1-2 が採点できないので，改めて Item2 として修正版を定義せよ．） 要するに，Item1 を Usable と Item2 に分離せよ．

1-4	問題設定		課題 1-3 までで開発の準備が整った．後はインスタンス化できるクラスの実装を行う．次の３つのクラスを実装せよ．クラス名欄の括弧書きを参照し，適切な extends と implements を実装せよ．extends と implements には Weapon，Item2，Usable のいずれかを用いることとする．
	クラス１	クラス名	Sword　（武器：剣）
		コンストラクタ	１．全てのフィールドの初期値を設定する．
		メソッド	attackEffect()を実装する． 処理　：「グサッ」とコンソール出力する．
	クラス２	クラス名	Cane　（武器：杖，　道具としても使用可能）
		コンストラクタ	１．全てのフィールドの初期値を設定する．
		メソッド	attackEffect()を実装する． 処理　：「ゴスッ」とコンソール出力する． useEffect()を実装する． 処理　：「○○を振りかざした」とコンソール出力する． ※○○には武器名が入る．
	クラス３	クラス名	Harb　（道具：やくそう，　道具として使用可能）
		コンストラクタ	１．全てのフィールドの初期値を設定する．
		メソッド	useEffect()を実装する． 処理　：「モシャモシャ」とコンソール出力する．

課題 2

2-1

問題設定

数秒悩み，ダジャレをコールバックする Joke クラスを誰かが開発したようだ．コールバックを受け取るための JokeListener インタフェースも準備されている．

joke.play()でダジャレを依頼し，ダジャレを待っている間，プログレスバーのように進捗が表示される図 1 の機能を実装したい．使い方は次の通りである．

1. Joke をインスタンス化する．コンストラクタには JokeListener を実装したインスタンスを引数とする．
2. Joke インスタンスに対し，int play()を実行する．play()は数秒間悩んだ後にダジャレをコールバックメソッド（void jokePlayed (String joke)）に送る．
3. int play()の戻り値は何秒悩むかを示している．

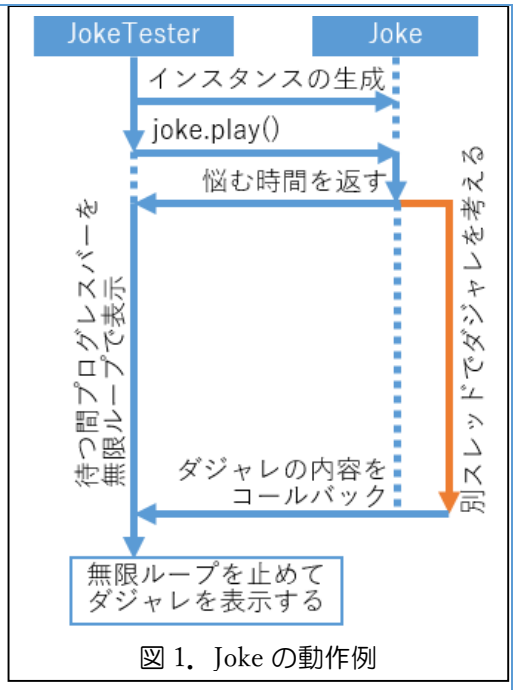

図 1．Joke の動作例

Joke を試すため以下の条件を満たすクラス JokeTester を開発せよ．Joke と JokeListener は Web から DL できる．

継承	なし	実装	JokeListener

フィールド
　joke: Joke 型のインスタンス変数
　flag: boolean 型（Joke のコールバックを待っていることを示すフラグ）

コンストラクタ
　引数なし
　処理：フィールド joke をインスタンス化する．

まずは，ここまでを実装したクラス JokeTester を開発せよ．Joke と JokeListener は Web から DL した後，eclipse のソースフォルダにドラッグアンドドロップでインポートするとよい．

<table>
<tr><td rowspan="4">2-2</td><td>メソッド</td><td>void jokeTest()：
Joke のテストを行う．
引数，戻り値：なし

処理（複雑なので手順を示す）
1. Joke を依頼するため joke.play()を実行し，戻り値（ダジャレ生成にかかる秒数）を得る（適当な変数に格納する）．
2. コールバックメソッド jokePlayed()が実行されるまでの間，0.5 秒で 1 ループする無限ループを実装する．0.5 秒待機する処理はヒントをコピペするとよい．無限ループ中には「ダジャレ待機中」とコンソール出力し，フリーズしているわけではないことを示してほしい．
3. コールバックメソッド jokePlayed()が実行されたら，無限ループが終わるように処理を修正する．boolean 型のフラグを用いるとよい．
4. 無限ループを止めると同時に，コールバックで送られてきたダジャレをコンソール出力し，処理を終了する．</td></tr>
<tr><td>ヒント</td><td>0.5 秒待つという処理は以下のように実装する．
try{
　　Thread.sleep(500)；　//500ms 待つ
}catch(Exception e){
}</td></tr>
<tr><td>テスト例</td><td>Main.java の main メソッドにて
JokeTester jt = new JokeTester()；
jt.jokeTest()；</td></tr>
<tr><td>出力例</td><td>ダジャレ待機中
ダジャレ待機中
ダジャレ待機中
ダジャレ待機中
フトンがフットンだ</td></tr>
</table>

<table>
<tr><td rowspan="4">2-3</td><td>メソッド</td><td>void jokeTestExp()：
Joke のテストを行う（発展）．
引数，戻り値：なし

処理（複雑なので手順を示す）
1. Joke を依頼するため joke.play()を実行し，戻り値（ダジャレ生成にかかる秒数）を得る（適当な変数に格納する）．
2. コールバックメソッド jokePlayed()が実行されるまでの間，0.5 秒で 1 ループする無限ループを実装する．0.5 秒待機する処理はヒントをコピペするとよい．無限ループ中にはプログレスバーのような表示をコンソール出力し，フリーズしているわけではないことを示してほしい．
3. コールバックメソッド jokePlayed()が実行されたら，無限ループが終わるように処理を修正する．boolean 型のフラグを用いるとよい．
4. 無限ループを止めると同時に，コールバックで送られてきたダジャレをコンソール出力し，処理を終了する．

プログレスバーのような表示とは，テスト例のように，0.5 秒毎に■の数が増えていく．したがって■と□の合計数はダジャレを待つ時間の 2 倍となる（テスト例は 4 秒待機の例）．なお括弧は半角，■と□は「しかく」を日本語変換してでてくる記号である．
【参考】長い処理がかかる Web 通信のような処理を実装する際，シングルスレッドで Web 通信を行うと，利用者はフリーズしたように感じてしまう．今回のようにマルチスレッド＋コールバックによってプログレスバーを実現してあげることでユーザビリティの向上につながる．</td></tr>
<tr><td>ヒント</td><td>0.5 秒待つという処理は以下のように実装する．
try{
　　Thread.sleep(500)；　//500ms 待つ
}catch(Exception e){
}</td></tr>
<tr><td>テスト例</td><td>Main.java の main メソッドにて
JokeTester jt = new JokeTester()；
jt.jokeTestExp()；</td></tr>
<tr><td>出力例</td><td>[■□□□□□□□]
[■■□□□□□□]
[■■■□□□□□]
[■■■■□□□□]
[■■■■■□□□]
[■■■■■■□□]
[■■■■■■■□]
[■■■■■■■■]
フトンがフットンだ</td></tr>
</table>