Chapter 4

継承、インターフェイス、パッケージ

以下の指示に従って、クラス、オブジェクトを作成・実行せよ。なお、文中では個別に指示しないが、変更を行ったら適宜コンパイルすること。

4.1 クラスの継承

- 次のChokinクラスを作成せよ。（次以降の手順ではあらかじめコンパイルされている必要がある）

```
class Chokin{
        int okane=0;
        Chokin(){
           okane=10000;
        }
        void setOkane(int i){
           okane+=i;
        }
        int getOkane(){
           return okane;
        }
}
```

- Chokinクラスを継承しChokinBakoクラスを作成せよ。ただし、ChokinBakoク

ラスにはboolean型の変数available（初期値はtrue）がフィールドとして含まれることとし、変数availableの値を返すisAvailableメソッドを追加せよ。さらにgetOkaneメソッドが呼び出されるとavailableをfalseにセットするよう再定義（オーバーライド）せよ。（Chokinクラスをコピー＆ペースとして処理を追加・・などとはしてはダメ）この際、コンストラクタは明示的に作らないものとする。

- Test04Aクラスを作成し、次の処理を行うmainメソッドを作成し、実行せよ。なお、継承されたクラスのコンストラクタは自動的に親クラスのコンストラクタを処理の最初に呼び出すことを思い出せ。「ChokinBakoクラスのオブジェクトkobutaを作成する。次に、kobutaオブジェクトのsetOkaneメソッドを利用してokaneフィールドに値 5000を追加せよ。その後、getOkaneメソッドで現在のokaneの値を取得し、isAbailableメソッドを用いてavailableの値を取得する。最後にこれら二つの値を画面に出力せよ」

- Chokinクラスのコンストラクタとして以下のものを追加せよ。

```
Chokin(int okane){
   this.okane=okane;
}
```

- ChokinBakoクラスのコンストラクタとして以下のものを追加せよ。

```
ChokinBako(int okane){
   super(okane);
}
```

- Test04Bクラスのmainメソッドを、kobutaオブジェクトの生成時に引数に50000を入れるコンストラクタを呼び出すように変更し実行せよ。

4.2　クラスの継承とコンストラクタの呼び出し

次のクラスCを実行し、得られた結果について考察せよ。

```
class A {
          A(){
              System.out.println("A");
          }
}
class B extends A{
          B(){
              System.out.println("B");
          }
}
class C{
          public static void main(String[] args){
              B b= new B();
          }
}
```

4.3 インターフェイス

- 次の二つのインターフェイスを作成せよ。(Circle.javaファイルに二つのインターフェイスの定義を記述しコンパイルする)

```
interface Circle{
 void setR(double r);
 double getArea();
}
interface Painted{
 boolean isFilled();
}
```

- インターフェイスCircleとPaintedを実装するクラスPaintedCircleを作成し、実行せよ。但し、以下のフィールド、メソッドを備えているものとする。
  - フィールド double r=1.0 (半径に相当)
  - メソッド public void setR(double r)　フィールドrに引数の値をセットする処理を行うものとする。

– メソッド public double getArea()　ただし、半径rの値から円の面積を計算して返す処理を行うものとする。

– メソッド main　ただし、PaintedCircleクラスのオブジェクトpcを生成し、オブジェクトpcのsetRメソッドを用いて、半径を10.0にセットし、さらにgetAreaメソッドを用いて面積を画面に表示する

実は、この仕様ではPaintedCircleは正しく実行できない。その理由について考察せよ。

- クラスPaintedCircleにメソッド public boolean isFilled()を追加せよ。ただし、単にtrueを返す処理を行うものとする。再度PaintedCircleを実行せよ。

4.4　パッケージ

以下、Linux環境で実施しているものとする。

- まず、各自のホームディレクトリ直下にjava0000という名前のディレクトリを作成し、カレントディレクトリをjava0000に移動せよ。
- カレントディレクトリ直下にcomという名前のディレクトリを作成せよ。次に、ディレクトリcomの中にshibauraという名前のディレクトリを作成し、さらにディレクトリshibaura内にディレクトリtoolsおよびディレクトリsystemを作成せよ。
- 次のプログラムをCar.javaとして作成し、コンパイルして出来たファイルCar.classを上記のディレクトリtoolsに配置せよ[1]。(クラスとメソッドにpublicを忘れずにつけること)

```
package  com.shibaura.tools;

public class Car{
        double speed=0.0;
```

[1]実は、予めコンパイルせずともソースファイルをパッケージの対応するディレクトリに置いておけばこれを利用するクラスをコンパイルするときに同時にコンパイルはされるのだが、classファイルがパッケージの該当部分に存在することが本質的に大事なので、本演習では予めコンパイルしてもらう。

```
        public void setSpeed(double ds){
              speed=ds;
        }
        public double getSpeed(){
              return speed;
        }
}
```

- 次のプログラムをCar.javaとして作成し、コンパイルして出来たファイルCar.classを上記のディレクトリsystemに配置せよ。(クラスとメソッドにpublicを忘れずにつけること)

```
package  com.shibaura.system;

public class Car{
        double speed=0.0;
        public void setSpeed(double ds){
              speed=ds;
        }
        public double getSpeed(){
              return speed*3600.0/1000.0; // converted to (km/h)
        }
}
```

- 次のプログラムをTest04C.javaとして作成し、ディレクトリjava0000直下に置いてコンパイルし、実行せよ。また、com.shibaura.systemパッケージのCarクラスを用いる場合はどうしたらよいか。適切に修正して実行せよ。

```
    import com.shibaura.tools.*;

    class Test04C{
        public static void main(String[] args){
                Car myCar = new Car();
                myCar.setSpeed(100.0);
                System.out.println("velocity:"+myCar.getSpeed());
```

```
        }
    }
```