Design Heuristics

for using Inheritance in Object-Oriented Systems

Favor composition over inheritance

```
public class Apple {

}

public class Fruit {

    public int peel() {

        System.out.println("Peeling is appealing.");

        return 1;

    }

}
```

```
public class Apple extends Fruit {

}

public class Fruit {

    public int peel() {

        System.out.println("Peeling is appealing.");

        return 1;

    }

}
```

```
public class Apple {

        private Fruit fruit = new Fruit();

        public int peel() {

                return fruit.peel();

        }

}
```

```
public class Fruit {

        public int peel() {

                System.out.println("Peeling is appealing.");

                return 1;

        }

}
```

```
public class Apple implements IPeelable {

    public int peel() {

        System.out.println("Peeling is appealing.");

        return 1;

    }

}

public interface IPeelable {

    int peel();

}
```

Inheritance should be used only to model a specialisation hierarchy

```
public class Dog {

        private String name;


        private Dog(String name) {

                this.name = name;

        }


        public void print() {

                System.out.println(name);

        }

}
```

```
public class Human extends Dog {


}
```

```
public class Rectangle {
        private int width;
        private int height;

        public void setWidth(int width) {
                this.width = width;
        }

        public void setHeight(int height) {
                this.height = height;
        }
}
```

```
public class Square extends Rectangle {

        public void setWidth(int width) {
                this.width  = width;
                this.height = width;
        }

        public void setHeight(int height) {
                this.height = height;
                this.width  = height;
        }
}
```

```
Rectangle r = new Square();
r.setWidth(10);
r.setHeight(5); // oops!
```

Base classes should be abstract

Prefer interfaces to abstract classes

Prefer class hierarchies to tagged classes

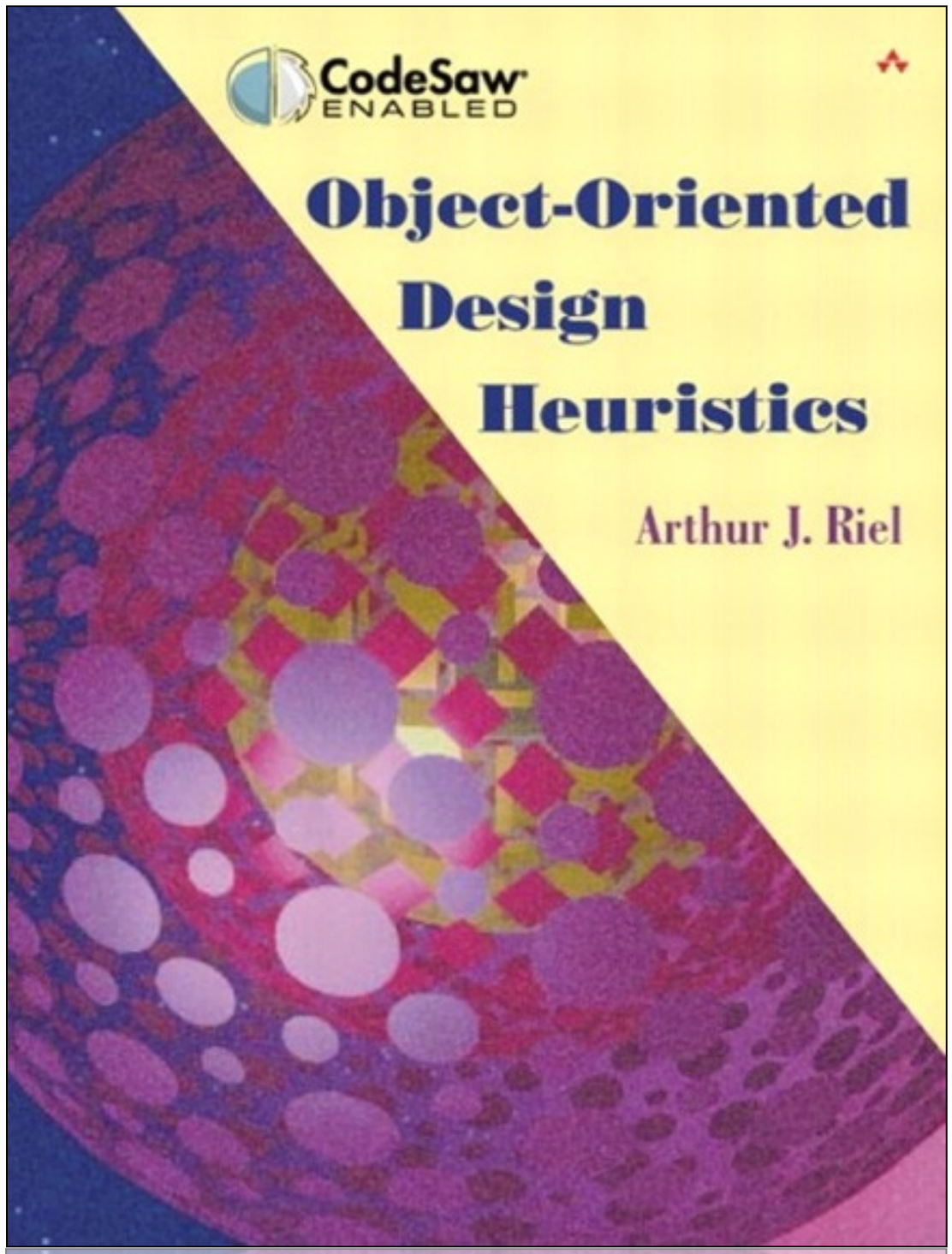
```
public class Thing {

    private String type;

    // constructor(s), accessors and behaviour

    public void makeNoise() {
        if(type.equals("Human")) {
            System.out.println("Lalala...");
        } else if(type.equals("Dog")) {
            System.out.println("Wuff, wuff...");
        } else if(type.equals("Car")) {
            System.out.println("Vroooom...");
        }
    }

}
```

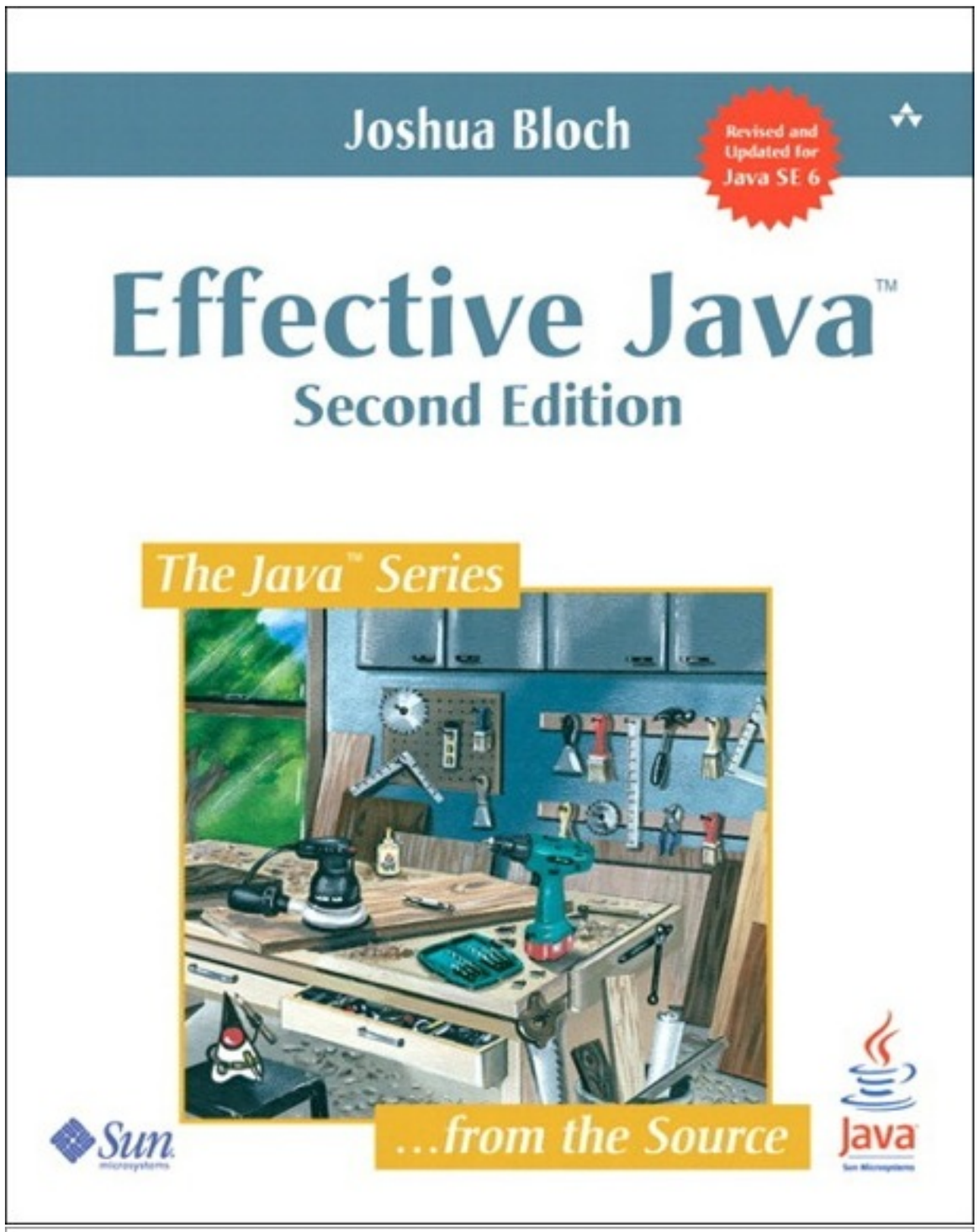
Do not turn objects of a class into derived classes of that class

```
public class Person {

    private String name;

    public Person(String name) {
        this.name = name;
    }

    // accessors and behaviour

}
```

```
public class Michael extends Person {

    public Michael() {
        super("Michael");
    }

}

public class Sebi extends Person {

    public Sebi() {
        super("Sebastian");
    }

}
```

If two or more classes share only common data (no common behavior), then the common data should be placed in a class which will be contained by each sharing class.


CodeSaw
ENABLED
Object-Oriented
Design
Heuristics
Arthur J. Riel


Joshua Bloch
Revised and
Updated for
Java SE 6
Effective Java™
Second Edition
The Java™ Series
Sun
microsystems
...from the Source
Java
Sun Microsystems