# JAVA Programming Midterm

Student Id: name: date:

✓ **Homework Part (60%)**

1. Which of the following techniques can be used to prevent the instantiation of a class by any code outside of the class? (3%)

A. Declare all constructors using the private access modifier.

B. Do not include a return statement in the constructor.

C. Do not declare any constructors inside a class definition.

D. Declare all constructors with a void return type.

E. None of the above.

2. Which one statement is true? (3%)

A. A final method in class X can be abstract if and only if X is abstract.

B. A protected method in class X can not be overridden by any subclass of X.

C. A private static method can be called only within other static methods in class X.

D. A non-static public final method in class X can be overridden in any subclass of X.

E. A public static method in class X can be called by a subclass of X without explicitly referencing the class X.

3. Given the following Java code: (3%)

Exhibit:

```
1.    public class SimpleCalc {
2.       public int value;
3.       public void calculate( ) { value += 7; }
4.    }

And:
5.    public class MultiCalc extends SimpleCalc {
6.       public void calculate( ) { value += 3; }
7.       public void calculate( int multipier) {
8.          calculate( );
9.          super.calculate( );
10.         value *= multipier;
```

```
11.         }
12.         public static void main(String[] args) {
13.             MultiCalc calculator = new MultiCalc( );
14.             calculator.calculate(2);
15.             System.out.println(“ Value is: “ + calculator.value);
16.         }
17.     }
```

What is the result?

A. Value is: 8

B. Compilation fails.

C. Value is: 20

D. Value is: -20

E. The code runs with no output.

4. Which of the following are legal identifiers? (3%)

A. _try

B. #try

C. try

D. *try

E. 1try

5. Given the following Java code: (3%)

```
1.      interface Count {
2.          short counter = 0;
3.          void countUp( );
4.      }
5.      public class TestCount implements Count {
6.
7.          public static void main(String[] args) {
8.              TestCount t = new TestCount( );
9.              t.countUp( );
10.         }
11.         public void countUp( ) {
```

```
12.             for (int x = 6; x > counter; x--, ++counter) {
13.                 System.out.println(“ “ + counter);
14.             }
15.         }
16.     }
```

What is the result?

A. 1 2 3

B. 0 1 2 3

C. 1 2 3 4

D. Compiler error. Because the variable type of counter is final static.

E. Compiler error. The method countUp( ) has no implementation in interface Count.

6. Given the following Java code: (3%)

```
1.          public class foo {
2.              public static void main (String[] args) {
3.                  String s;
4.                  System.out.println(“s=” + s);
5.              }
6.          }
```

What is the result?

A. The code compiles and “s=” is printed.

B. The code compiles and “s=null” is printed.

C. The code compiles and “s=0” is printed.

D. The code does not compile because string s cannot be referenced.

E. The code does not compile because string s is not initialized.

F. There is a runtime error.

7. Given the following Java code: (3%)

```
1.          public class HorseTest {
2.              public static void main (String[] args) {
3.                  class Horse {
```

```
4.                public String name;
5.                public Horse(String s) {
6.                   name = s;
7.                }
8.             }
9.          Object obj = new Horse("Zippo");
10.            Horse h = (Horse) obj;
11.            System.out.println(h.name);
12.         }
13.      }
```

What is the result?

A. Compilation Error at line 3

B. Compilation Error at line 9

C. Compilation Error at line 10

D. Compilation Error at line 11

E. None of the above.

8. Given the following Java code: (3%)

```
1.       public class TestObj {
2.          public static void main (String [] args) {
3.             Object o = new Object() {
4.                public boolean equals(Object obj) {
5.                   return true;
6.                }
7.             };
8.             System.out.println(o.equals("Fred"));
9.          }
10.      }
```

What is the result?

A. true

B. Compilation fails because of an error on line 3.

C. Compilation fails because of an error on line 4.

D. Compilation fails because of an error on line 8.

E. Compilation fails because of an error on a line other than 3, 4, or 8.

9. Given the following Java code: (3%)

```
class SomeException:
1.      public class SomeException {
2.      }

class A:
1.      public class A {
2.          public void doSomething() throws Exception {}
3.      }

class B:
1.      public vlass B extends A {
2.          public void soSomething() throws Exception {}
3.      }
```

Which statement is true about the two classes?

A. Compilation of both classes will fail.

B. Compilation of both classes will succeed.

C. Compilation of class A will fail, Compilation of class B will succeed.

D. Compilation of class B will fail, Compilation of class A will succeed.

10. Given the following Java code: (3%)

```
1.      class A {
2.          public static void main(String[] args){
3.              int a=0, b=5;
4.              String c[] = {“A”, “B”, “C”};
5.              try {
6.                  System.out.print(c[a/b]);
7.                  try {
8.                      for(int i=1; i<4; i++) {
9.                          System.out.print(c[i]);
10.                     }
11.                 }
12.                 catch (Exception e)
13.                 {
14.                     System.out.println(“D”);
```

```
15.                 }
16.                 finally {
17.                     System.out.println("E");
18.                 }
19.             }
20.             catch (Exception e) {
21.                 System.out.println("F");
22.             }
23.             finally {
24.                 System.out.println("G");
25.             }
26.         }
27.     }
```

What is the result?

A. Prints: AABCG

B. Prints: ABCDG

C. Prints: AABCDG

D. Prints: ABCDEG

E. Prints: ABCDEFG

11. Given the following Java code: (3%)

```
1.      class B extends Exception {}
2.      class C extends B {}
3.      class D extends C {}
4.      class A {
5.          public static void main(String args[]) {
6.              int a,b,c,d,x,y,z;
7.              a = b = c = d = x = y = 0;
8.              z = 1;
9.              try {
10.                 try {
11.                     switch(z) {
12.                         case 1: throw new B();
13.                         case 2: throw new C();
14.                         case 3: throw new D();
15.                         case 4: throw new Exception();
```

```
16.                 }
17.                 a++;
18.             }
19.             catch ( C e ) {b++;}
20.             finally{c++;}
21.         }
22.         catch ( B e ) {d++;}
23.         catch ( Exception e ) {x++;}
24.         finally {y++;}
25.         System.out.print(a+","+b+","+c+","+d+","+x+","+y);
26.       }
27.    }
```

What is the result?

A. 0,0,1,1,0,0

B. 0,1,0,1,1,0

C. 0,0,1,1,0,1

D. 0,1,1,1,1,1

E. 1,1,0,1,0,0

12. Given the following Java code: (3%)

```
1.  class Animal { public String noise () { return "peep"} }
2.     class Dog extends Animal {
3.        public String noise () { return "back"; }
4.  }
5.  class Cat extends Animal {
6.     public String noise () { return "move"; }
7.  }
8.  . . . //skip parts of code
9.  Animal animal = new Dog();
10. Cat cat = ( Cat ) animal;
11. System.out.printIn( cat.noise() ) ;
```

What is the result?

A. peep

B. back

C. move

D. Compilation fails.

E. An exception is thrown at runtime

13. Given the following Java code: (3%)

```
1.  class Pizza {
2.      java.util.ArrayList toppings;
3.      public final void addTopping(String topping) {
4.          toppings.add(topping);
5.      }
6.  }
7.  public class PepperoniPizza extends Pizza {
8.      public void addTopping(String topping) {
9.          System.out.println("Cannot and Uoppings");
10.     }
11.     public static void main(String[] args) {
12.         Pizza pizza = new PepperoniPizza();
13.         Pizza.addTopping("Mushrooms");
14.     }
15. }
```

What is the result ?

A. Cannot and Uoppings

B. Compilation fails

C. The code runs with no output

D. A NullPointerException is thrown in Line 4

E. None of above.

14. Given the following Java code: (3%)

```
1.  interface Foo {}
2.  class Alpha implements Foo {}
3.  class Beta extends Alpha {}
4.  class Delta extends Beta {
5.      public static void main(String[] args) {
6.          Beta x = new Beta ();
7.          // insert code here
8.      }
9.  }
```

Which code, inserted at line 7 will cause a java.lang.ClassCastException?

A. Alpha a = x;

B. Foo f = (Delta)x;

C. Foo f = (Alpha)x;

D. Beta b = (Beta)(Alpha)x;

15. Given the following Java code: (3%)

```
1.    class C {
2.        public static void main(String[] args) {
3.            int a=0, b=5;
4.            try {
5.                System.out.print(a/b+b/a);
6.            } catch {
7.                System.out.println("Exceptions!!!");
8.            }
9.        }
10.   }
```

What is the result of attempting to compile the program?

A. Compiler Error

B. Prints Nothing

C. Prints: Exceptions!!!

D. Runtime Error

E. None of the above

16. Given the following Java code: (3%)

```
1.    class A {
2.        public static void main (String[] args) {
3.            byte a[] = new byte[2];
4.            long b[] = new long[2];
5.            float c[] = new float[2];
6.            Object d[] = new Object[2];
7.            System.out.print(a[1]+","+b[1]+","+c[1]+","+d[1]);
8.        }
9.    }
```

What is the result?

A. Prints: 0,0,0,null

B. Prints: 0,0,0.0,null

C. Prints: 0,0,0,0

D. Prints: null,null,null,null

E. The code runs with no output.

17. Given the following Java code: (3%)

```
1.    class A {
2.        static void my() throws ArithmeticException {
3.            System.out.print("A");
4.            throw new ArithmeticException("A");
5.        }
6.        public static void main (String args []) {
7.            try {
8.                my();
9.            }
10.           catch (Exception e) {
11.               System.out.print("B");
12.           }
13.           finally {
14.               System.out.print("C");
15.           }
16.       }
17.   }
```

What is the result?

A. Prints: A

B. Prints: C

C. Prints: AC

D. Prints: ABC

E. Prints: AABC

18. Given the following Java code: (3%)

```
1.  interface Foo {}
2.  class Alpha implements Foo {}
3.  class Beta extends Alpha {}
4.  class Delta extends Beta {
5.      public static void main(String[] args) {
6.          Beta x = new Beta ();
7.          // insert code here
8.      }
9.  }
```

Which code, inserted at line 7 will cause a java.lang.ClassCastException?

A. Alpha a = x;

B. Beta b = (Beta)(Alpha)x;

C. Foo f = (Delta)x;

D. Foo f = (Alpha)x;

19. Given the following Java code: (3%)

```
1.      class A {
2.          public static final int a = 1;
3.          protected static int b = 2;
4.          int c = 3;
5.          static class B {
6.              int d = a;
7.              int e = b;
8.              int f = c;
9.          }
10.         class C {
11.             int g = a;
12.             int h = b;
13.             int I = c;
14.         }
15.     }
```

What is the result?

A. Compilation Error at line 2.
B. Compilation Error at line 8.
C. Compilation Error at line 10.
D. Run without any problem.
E. An exception is thrown at runtime.

20. Given the following Java code: (3%)

```
1.      class B {
2.          private int x = 2;
3.          static A a1 = new A(2,1) {
4.              public A(int tmp) {x(tmp); y(tmp);};
5.              public int m() {return x()+y();}
6.          };
7.          public static void main(String[] args) {
8.              System.out.print(a1.m());
9.          }
10.     }
11.     abstract class A {
12.         private int x = 4;
13.         private int y = 2;
14.         private int z = 6;
15.         public int x() {return x;}
16.         public void x(int x) {this.x = x;}
17.         public int y() {return y;}
18.         public void y(int y) {this.y = y;}
19.         public abstract int m();
20.     }
```

What is the result?

A. Prints: 8
B. Prints: 3122
C. Compilation fails
D. Run-time error
E. None of the above

✓ **Lab1 Part: Template (Method) Pattern (20%)**

*Definition: Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.* (See Figure 1 for its UML class diagram and note that *italic method* means it is abstract method.)

Actually, we are using *Template Method Pattern* in lab1. That is, we can use a method before having its real implementation.

**21. Please point out where lab1 uses Template Pattern. (5%)**

Next, let's check the following question:

***A company wants to produce two kinds of drink machines. One is for tea and the other is for coffee. They find that the procedures of two machines are similar as following. Figure 2 shows their structures.***

**22. What's the drawback of the codes in Figure 2? (5%)**

**23. Please draw the class diagram for the drink machine after applying Template Pattern to this problem to eliminate the drawback you claim. You can refer to lab1 to design the class diagram. (10%)**

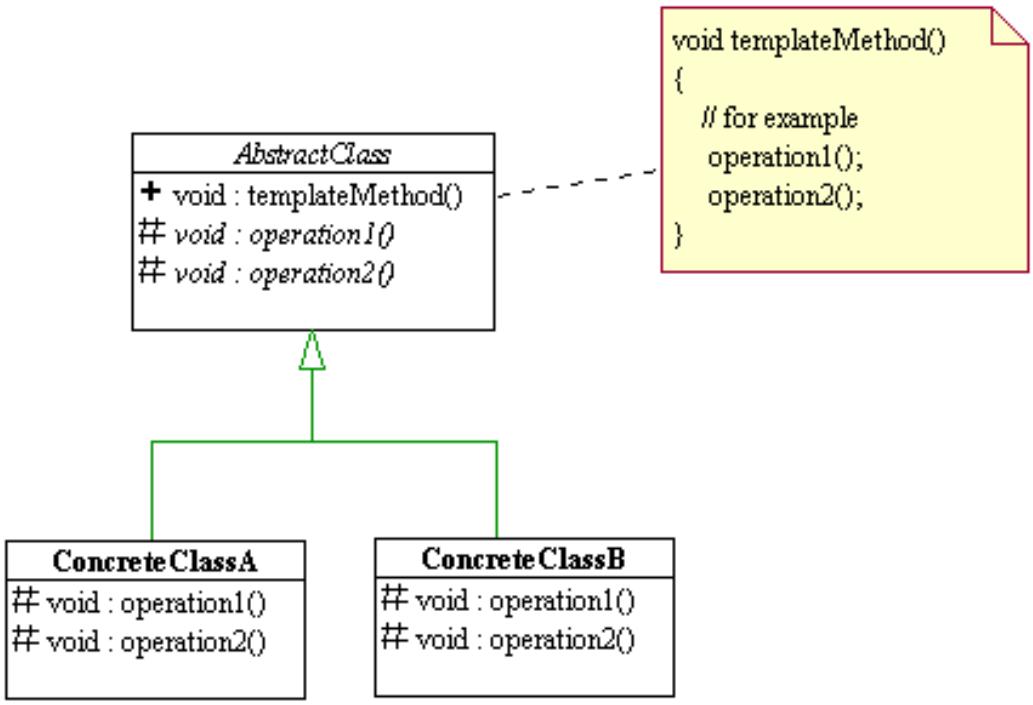


Figure 1

```
class Drink {
    //ignore
}
class Tea extends Drink {
    //ignore
}
class Coffee extends Drink {
    //ignore
}
```

```
class TeaMachine{
    Drink tea;
    void boilWater(){
        //boil water
    }
    void makeTea(){
        //steep tea in boiling water
    }
    void pourInCup(){
        //pour tea in cup
    }
    void addFlavoring(){
        //add lemon
    }
    Drink getDrink(){
        boilWater();
        makeTea();
        pourInCup();
        addFlavoring();
        return tea;
    }
}
```

```
class CoffeeMachine{
    Drink coffee;
    void boilWater(){
        //boil water
    }
    void makeCoffee(){
        //brew coffee in boiling water
    }
    void pourInCup(){
        //pour coffee in cup
    }
    void addFlavoring(){
        //add sugar and milk
    }
    Drink getDrink() {
        boilWater();
        makeCoffee();
        pourInCup();
        addFlavoring();
        return coffee;
    }
}
```

Figure 2

✓ **Advanced (30%)**

- **String problem (2%)**

  **24.** The method "*substring*" in *java.lang.String* could result in *memory leak* problem. Please use a short example to show the problem and provide a solution by using the constructor "*String(String original)*".

- **To be (inherited) or not to be (inherited)? That is the question! (2%)**

  **25.** See Fig 3. Should *Rectangle* be extended by *Square?* Please explain it. (Hint: By applying *Liskov Substitution Principle*).

- **Call by XXX problems (6%)**

  **26.** Someone says that Java can write a method for swapping two values when the parameter is an *int array*. That is, swapping two values inside an array. Is it correct? Why? (2%)

  **27.** See Fig.4. According the above argument. A guy proposes a program for swapping variables, i.e., variables *a* and *b*. Is it correct? Why? (2%)

  **28.** See Fig.5. Another guy says that *primitive types* cannot be swapped, and claims that we should use *wrappers*. He writes a program as Fig.5. Is it correct? Why? (2%)

- **Overriding problem (6%)**

  **29.** See Fig.6. What's the output? Note that if you give a wrong answer, you will lose 1 point until there are no points to lose.

- **Human compiler is more powerful (4%)**

  **30.** Please see Fig.7. This class is designed for describing a student's name and score. There are 2 *compile errors* and 2 *design mistakes.* Please point them out and correct them. Note that *design mistake* means that it can be compiled successfully; however, it has a *conceptual error*.

  - Note also that if you treat a correct one as a wrong one, you will lose 1 point until there are no points to lose.

- **Challenge (10%)**

  - *Warning: Please do not try this if you haven't finished the questions above.*
  - There is a design pattern called <u>*Decorator Pattern*</u>. Please see Fig.8 for its simplified class diagram and some comments. In short, decorator pattern attaches additional responsibilities to an object dynamically.

    **31.** Why does *Decorator* extend *Component*? (4%)

    **32.** Both *ConcreteDecoratorA* and *ConcreteDecoratorB* have an instance member called *wrappedObj*. What *wrappedObj* for? (3%)

    **33.** This pattern can attach additional responsibility to an existing design. However, inheritance can also attach new functionality. What's the difference? (3%)

```
class Square extends Rectangle {
	public void setHeight(double height) {
		super.setHeight(height);
		super.setWidth(height);
	}

	public void setWidth(double width) {
		super.setHeight(width);
		super.setWidth(width);
	}
}

class Rectangle {
	double width;
	double height;
	public double getHeight() {
		return height;
	}
	public void setHeight(double height) {
		this.height = height;
	}
	public double getWidth() {
		return width;
	}
	public void setWidth(double width) {
		this.width = width;
	}
}
```

Fig. 3

```
class Swapper {
	public static void swap(int[] array) {
		int tmp=array[0];
		array[0]=array[1];
		array[1]=tmp;
	}
	public static void main(String[] args) {
		int a=0, b=1;
		int[] array = new int[2];
		array[0]=a;
		array[1]=b;
		swap(array);
	}
}
```

Fig. 4

```
class Swapper {
	public static void swap(Integer a, Integer b) {
		Integer tmp = a;
		Integer a=b;
		Integer b=tmp;
	}
	public static void main(String[] args) {
		Integer a=new Integer(0);
		Integer b=new Integer(1);
		swap(a,b);
	}
}
```

Fig. 5

```
class SuperClass {
	int x = 5;
	static int y = 10;
	public static void staticMethod() {
		System.out.println(y);
	}
	public void instanceMethod() {
		System.out.println(x);
	}
}
public class Test extends SuperClass {
	int x = 15;
	static int y = 20;
	public static void staticMethod() {
		System.out.println(y);
	}
	public void instanceMethod() {
		System.out.println(x);
	}
	public static void main(String[] args) {
		SuperClass obj = new Test();
		obj.staticMethod();
		( (Test)obj ).staticMethod();

		obj.instanceMethod();
		( (Test)obj ).instanceMethod();

		System.out.println( obj.x );
		System.out.println( ( (Test)obj ).x);
	}
}
```

Fig. 6

```
protected class ErrorStudentClass extends Object {
	private static String name;
	private static int score;
	private ErrorStudentClass(String name, int score) {
		super();
		this.name = name;
		if(score < 0 || score > 100) {
			throw new Exception("Error Score");
		}
		else {
			this.score = score;
		}
	}
	private ErrorStudentClass(int score, String name) {
		this(name,score);
	}
	public boolean isPassed() {
		if(score >= 60) {
			return true;
		}
		return false;
	}
}
```

Fig. 7

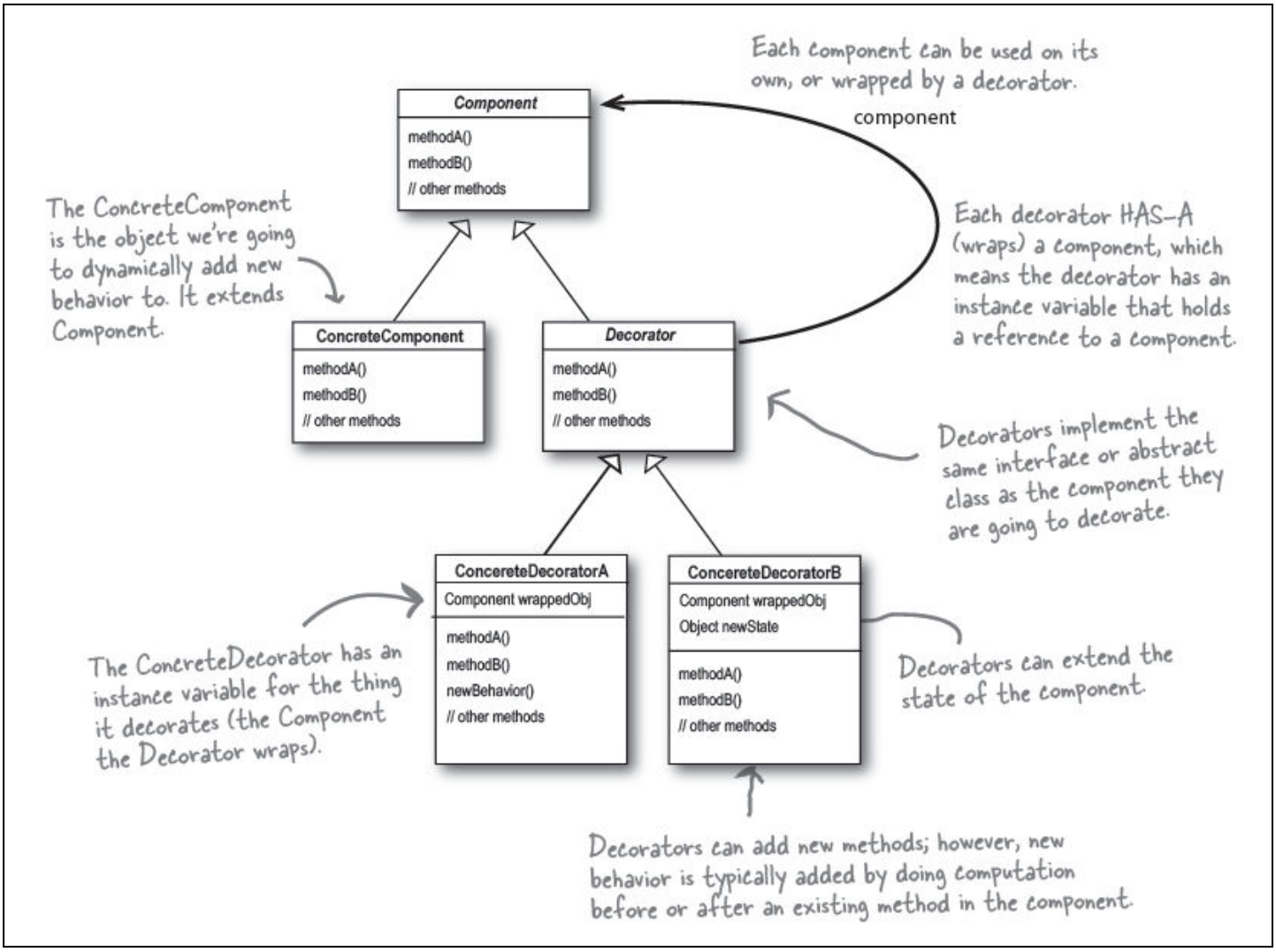
Each component can be used on its own, or wrapped by a decorator.
Component
methodA()
methodB()
// other methods
component
The ConcreteComponent is the object we're going to dynamically add new behavior to. It extends Component.
Each decorator HAS-A (wraps) a component, which means the decorator has an instance variable that holds a reference to a component.
ConcreteComponent
methodA()
methodB()
// other methods
Decorator
methodA()
methodB()
// other methods
Decorators implement the same interface or abstract class as the component they are going to decorate.
ConcereteDecoratorA
Component wrappedObj
methodA()
methodB()
newBehavior()
// other methods
ConcereteDecoratorB
Component wrappedObj
Object newState
methodA()
methodB()
// other methods
The ConcreteDecorator has an instance variable for the thing it decorates (the Component the Decorator wraps).
Decorators can extend the state of the component.
Decorators can add new methods; however, new behavior is typically added by doing computation before or after an existing method in the component.

Fig.8