

```
// DataStructures&AlgorithmsInJava (Adam Drozdek, 4th ed.)  
// Instructor's Solutions Manual  
// © 2013 Cengage Learning, all rights reserved.
```

1

OBJECT-ORIENTED PROGRAMMING USING JAVA

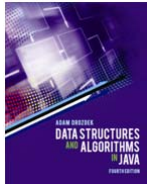
1. Constructors do not have a return type.
2. The following table indicates different accessing modes created with the reserved words `private`, `protected`, and `public`. The rows indicate the place from which an access is made and show whether the access is possible for a particular declaration.

Place of access	<code>private</code>	No modifier	<code>protected</code>	<code>public</code>
Same class	yes	yes	yes	yes
same package subclass	no	yes	yes	yes
Same package non-subclass	no	yes	yes	yes
Different package subclass	no	no	yes	yes
Different package non-subclass	no	no	no	yes

3. Calls `object1.process1(1000)` and `object1.process4(2000)` refer to methods defined in `ExtC`, because `object1` is an instance of this class.

`process1()` defined in `ExtC` does not override `process1()` in `C`, because their signatures are different. Therefore, because `object2` is of type `C`, the system looks for a definition of `process1(int)` in `C`. It does not find one, thus, a compilation error is issued, to the effect that an explicit cast is needed, as in `object2.process1((char)3000)`, in which case the character with Unicode value 3000 should be printed by `process1()` defined in `C`. For a similar reason, the statement `object2.process4(4000)` also causes a compilation error, because `process4()` is not defined in `C`. Note that `object2` is assigned an object of type `ExtC`, which may suggest that `process4()` defined in `ExtC` should be called, however, the compilation error is caused by the type of `object2`, which is `C`, not `ExtC`. To have the benefit of polymorphism, the method in `ExtC` must have exactly the same signature as a corresponding method in `C`.

`object3.process1('P')` calls `process1()` defined in `C`.



```
// DataStructures&AlgorithmsInJava (Adam Drozdek, 4th ed.)  
// Instructor's Solutions Manual  
// © 2013 Cengage Learning, all rights reserved.
```

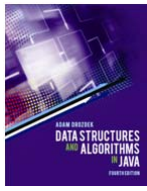
object3.process2('Q') calls process2() defined in ExtC on account of dynamic binding. Finally, object3.process3('R') invokes method process3() defined in C however, process3() calls process2(), and because object3 refers to an instance of ExtC, then, due to dynamic binding, process3() calls process2() defined in ExtC.

4. (a)

```
interface I2 extends I1 {  
    double I2f1();  
    void I2f2(int i);  
    // int I1f1(); - this can be done, but it is redundant;  
    // double I2f1() { return 10; } - it cannot have a body;  
    // private int AC1f4(); - interface methods cannot be  
    // private, static, synchronized, final;  
    // private int n = 10; - interface fields cannot be private or  
    // protected;  
}
```
- (b)

```
class CI1 implements I1 {  
    // int I1f1() { . . . . . } - must be public;  
    // void I1f2(int i) { . . . . . } - must be public;  
    int CI1f3() { . . . . . }  
}
```
- (c) An interface cannot be instantiated.
5. (a)

```
abstract class AC1 {  
    int AC1f1() { . . . . . }  
    void AC1f2(int i) { . . . . . }  
    // int AC1f3(); - a method without body must be declared  
    // abstract, as in: abstract int AC1f3();  
}
```
- (b) CAC1 must be an interface.
- (c) Multiple inheritance is not supported.
- (d) Abstract class cannot be instantiated.



```
// DataStructures&AlgorithmsInJava (Adam Drozdek, 4th ed.)  
// Instructor's Solutions Manual  
// © 2013 Cengage Learning, all rights reserved.
```

6. Definition

```
public boolean equals(SomeInfo si) {  
    return n == si.n;  
}
```

does not override the definition

```
public boolean equals(Object si) {  
    return n == ((SomeInfo) si).n;  
}
```

because the two methods have different signatures; to override the standard method `equals()`, the parameter must be of type `Object`.