DATA ENCAPSULATION

In our natural world, an object protects its delicate insides by surrounding it in a tough shell.

Example
Encapsulation in nature.

Hardware engineers use this same idea and seal the sensitive parts of the devices they build within a hard case.

Example
Encapsulation in hardware.

In software, ***data encapsulation*** is the practice of protecting the internal state of a computer object from tampering by other objects. The programmer "hides" the object's instance variables and provides public methods through which outsiders interact with the object.

Java enables you to practice data encapsulation by tagging class fields with the modifier **`private`**, which prevents other objects from tampering with them.

Example
Encapsulation in Java.

```
public class Capsule
{
   // 'content' cannot be touched by
   // any other object
   private int content;
   . . .
}
```

An ***interface*** is a point of contact through which two objects interact.

Example

Consider a digital wall thermostat used to control the heating and air conditioning of a room. Typically it has a display showing the temperature of the room and the preferred temperature setting. Both of these values are protected from any direct manipulation by the user. Buttons allow the user to step the preferred temperature setting up or down.

Computer objects can interact through any instance variable or method that has been made **`public`**.

Example

A Java class that models the HVAC thermostat is outlined below. The fields are made private to protect them from tampering by other objects in the system. Methods that perform the button actions are made public so that other objects in the system can call them. The public methods make up a **`Thermostat`** object's interface.

```
 1  // A Thermostat object models an HVAC thermostat.
 2  public class Thermostat
 3  {
 4     private int roomTemp; // current room temperature
 5     private int setTemp;  // preferred temp setting
 6     . . .
 7     public void up( )     // increase setTemp by 1 degree
 8     . . .
 9     public void down( )   // decrease setTemp by 1 degree
10     . . .
11  }
```

In Java, data encapsulation is enforced class-by-class at compile time (rather than object-by-object during runtime).

Example

Two classes are shown below: class **`Capsule`** and the application **`UseCapsule`**. These classes should be stored in files **`Capsule.java`** and **`UseCapsule.java`**, respectively. The field named **`content`** is tagged **`private`** at line 4. This restricts its access to methods within class **`Capsule`**. When the compiler encounters line 12 within class **`UseCapsule`**, it issues the diagnostic:

```
content has private access in Capsule
```

Capsule.java

```
 1  public class Capsule
 2  {
 3     // 'content' can't be touched by any other object
 4     private int content;
 5     . . .
 6  }
```

UseCapsule.java

```
 7  public class UseCapsule
 8  {
 9     public static void main( String [] args )
10     {
11        Capsule c = new Capsule( );
12        c.content = 0;  // ERROR
13        . . .
14     }
15  }
```

The programmer of class *A* sometimes needs to manipulate a private field in class *B*. He or she can do so if the programmer of class *B* has provided public get and set methods as part of *B*'s interface. A ***get method*** returns the value of an object's attribute; ***set method*** initializes it. Java programmers use a naming convention to allow you to quickly identify get and set methods – **`getX`** and **`setX`** where *X* is the name of the desired attribute.

Example

Class **`Capsule`** below has a get method for field **`content`** on lines 6–9 and a set method for it on lines 11–14. Both are part of a **`Capsule`** object's interface. With these methods in place, the application class **`UseCapsule`** can use them to initialize and inspect the private instance variable **`content`** within object **`cap`**, as shown on lines 21 and 22.

```
 1  public class Capsule
 2  {
 3     // 'content' can't be touched by any other object
 4     private int content;
 5
 6     public int getContent( )
 7     {
 8        return content;
 9     }
10
11     public void setContent( int c )
12     {
13        content = c;
14     }
15  }
```

```
16  public class UseCapsule
14  {
18     public static void main( String [] args )
19     {
20        Capsule cap = new Capsule( );
21        cap.setContent( 40 );
22        System.out.println( cap.getContent( ) );
23        . . .
24     }
25  }
```

When a field in a superclass is to be encapsulated from outside classes but must be accessible to the subclasses, you tag it **`protected`**.

Example

Field **`color`** (see line 3) is accessible to all classes in the **`WritingImplement`** hierarchy.
Field **`hasEraser`** (see line 15) is accessible to **`Pencil`** and **`MechanicalPencil`**.

```
 1  public class WritingImplement
 2  {
 3     protected Color color;
 4     . . .
 5  }
 6
 7  public class Pen extends WritingImplement
 8  {
 9     private double length;
10     . . .
11  }
12
13  public class Pencil extends WritingImplement
14  {
15     protected boolean hasEraser;
16     . . .
17  }
18
19  public class MechanicalPencil extends Pencil
20  {
21     private double leadWidth;
22     . . .
23  }
```

For each field in a class, four levels of encapsulation are possible – three by tagging the field with one of the keywords **`private`**, **`public`** or **`protected`** and the fourth by not tagging the field at all. The four possibilities are summarized in this table:

Access Tag	**Where the field (properly qualified) can be accessed**
`private`	Within the same class in which it is declared
none	Within all classes in the same package as the class in which it is declared
`protected`	Within all classes in the same package as the class in which it is declared and all subclasses of the class in which it is declared even if they are in different packages
`public`	Within all classes of the program

Exercises

Enter class **`Capsule`** (from page 4) into jGRASP and save it to a file. Compile it. Perform the following experiments using jGRASP's Interactions pane. To answer the questions, observe jGRASP's Workbench pane.

1. In jGRASP's Interactions pane, enter and execute the statement:

   ```
   Capsule c = new Capsule( );
   ```

   In jGRASP's Workbench pane, find the entry for **`c`** and click the so that it becomes  . What is the value of the object's instance variable **`content`**?

2. In jGRASP's Interactions pane, enter a statement that verifies that you cannot access the object's private instance variable **`content`**.

3. In jGRASP's Interactions pane, call the **`set`** method to set the object's instance variable to 20.

   In jGRASP's Workbench pane, verify that you did it correctly.

4. In jGRASP's Interactions pane, enter and execute a statement that prints the object's instance variable by calling its **`get`** method.