AP® COMPUTER SCIENCE A
2006 SCORING GUIDELINES

Question 2: Taxable Items (Design)

Part A: `purchasePrice` **2 1/2 points**

+1 call `getListPrice()`

+1 calculate correct purchase price (*no penalty if truncate/round to 2 decimal places*)

+1/2 return calculated price

Part B: `Vehicle` **6 1/2 points**

+1/2 `class Vehicle extends TaxableItem`
+1/2 `private double dealerCost`
+1/2 `private double dealerMarkup` (*no penalty if also store tax in field*)

+2 1/2 constructor
- **+1/2** `Vehicle(double ?, double ?, double ?)`
  `int/float` (*OK if match fields*)
- **+1** call parent constructor
  - **+1/2** attempt using `super`
  - **+1/2** correct call: `super(rate)` (*note: must be first line in method*)
- **+1** initialize dealer cost and markup fields
  - **+1/2** attempt (must use parameters on RHS or in mutator call)
  - **+1/2** correct

+1 `changeMarkup`
- **+1/2** `public void changeMarkup(double ?)`
  `int/float` (*OK if matches field; no penalty if returns reasonable value*)
- **+1/2** assign parameter to markup field

+1 1/2 `getListPrice`
- **+1** `public double getListPrice()`
- **+1/2** return sum of dealer cost and markup fields

Note: -1 usage if reimplement `purchasePrice` to do anything other than
`return super.purchasePrice();`

AP® COMPUTER SCIENCE A/AB
2006 GENERAL USAGE

Most common usage errors are addressed specifically in rubrics with points deducted in a manner other than indicated on this sheet. The rubric takes precedence.

Usage points can only be deducted if the part where it occurs has earned credit.

A usage error that occurs once when the same usage is correct two or more times can be regarded as an oversight and not penalized. If the usage error is the only instance, one of two, or occurs two or more times, then it should be penalized.

A particular usage error should be penalized only once in a problem, even if it occurs on different parts of a problem.

Nonpenalized Errors

spelling/case discrepancies*

local variable not declared when any other variables are declared in some part

default constructor called without parens; for example, `new Fish;`

use keyword as identifier

`[r,c]`, `(r)(c)` or `(r,c)` instead of `[r][c]`

= instead of == (and vice versa)

`length`/`size` confusion for array, `String`, and `ArrayList`, with or without `()`

`private` qualifier on local variable

extraneous code with no side-effect, for example a check for precondition

common mathematical symbols for operators (x • ÷ ≤ ≥ < > ≠)

missing { } where indentation clearly conveys intent

missing `( )` on method call or around `if/while` conditions

missing `;`s

missing "new" for constructor call once, when others are present in some part

missing downcast from collection

missing `int` cast when needed

missing `public` on class or constructor header

Minor Errors (1/2 point)

confused identifier (e.g., `len` for `length` or `left()` for `getLeft()`)

no local variables declared

`new` never used for constructor calls

`void` method or constructor returns a value

modifying a constant (`final`)

use `equals` or `compareTo` method on primitives, for example
`int x; …x.equals(val)`

`[]` – `get` confusion if access not tested in rubric

assignment dyslexia, for example, `x + 3 = y;` for `y = x + 3;`

`super(method())` instead of `super.method()`

formal parameter syntax (with type) in method call, e.g., `a = method(int x)`

missing `public` from method header when required

"false"/"true" or 0/1 for boolean values

"null" for `null`

Major Errors (1 point)

extraneous code which causes side-effect, for example, information written to output

use interface or class name instead of variable identifier, for example `Simulation.step()` instead of `sim.step()`

`aMethod(obj)` instead of `obj.aMethod()`

use of object reference that is incorrect, for example, use of `f.move()` inside method of `Fish` class

use private data or method when not accessible

destruction of data structure (e.g., by using root reference to a `TreeNode` for traversal of the tree)

use class name in place of `super` either in constructor or in method call

**Note: Spelling and case discrepancies for identifiers fall under the "nonpenalized" category as long as the correction can be unambiguously inferred from context. For example, "Queu" instead of "Queue". Likewise, if a student declares "Fish fish;", then uses Fish.move() instead of fish.move(), the context allows for the reader to assume the object instead of the class.*

AP® COMPUTER SCIENCE A
2006 CANONICAL SOLUTIONS

Question 2: Taxable Items (Design)

PART A:

```
public double purchasePrice()
{
  return (1 + taxRate) * getListPrice();
}
```

PART B:

```
public class Vehicle extends TaxableItem
{
  private double dealerCost;
  private double dealerMarkup;

  public Vehicle(double cost, double markup, double rate)
  {
    super(rate);
    dealerCost = cost;
    dealerMarkup = markup;
  }

  public void changeMarkup(double newMarkup)
  {
    dealerMarkup = newMarkup;
  }

  public double getListPrice()
  {
    return dealerCost + dealerMarkup;
  }

}
```

A2 A,

(a) Write the `TaxableItem` method `purchasePrice`. The purchase price of a `TaxableItem` is its list price plus the tax on the item. The tax is computed by multiplying the list price by the tax rate. For example, if the tax rate is 0.10 (representing 10%), the purchase price of an item with a list price of $6.50 would be $7.15.

Complete method `purchasePrice` below.

```
// returns the price of the item including the tax
public double purchasePrice()

    double taxTotal = getListPrice() * taxRate;

    return taxTotal + getListPrice();
```

Part (b) begins on page 12.

A2 A2

(b) Create the `Vehicle` class, which extends the `TaxableItem` class. A vehicle has two parts to its list price: a dealer cost and dealer markup. The list price of a vehicle is the sum of the dealer cost and the dealer markup.

For example, if a vehicle has a dealer cost of $20,000.00, a dealer markup of $2,500.00, and a tax rate of 0.10, then the list price of the vehicle would be $22,500.00 and the purchase price (including tax) would be $24,750.00. If the dealer markup were changed to $1,000.00, then the list price of the vehicle would be $21,000.00 and the purchase price would be $23,100.00.

Your class should have a constructor that takes dealer cost, the dealer markup, and the tax rate as parameters. Provide any private instance variables needed and implement all necessary methods. Also provide a public method `changeMarkup`, which changes the dealer markup to the value of its parameter.

```
public class Vehicle extends TaxableItem {
    private double listPrice;
    private double dealerCost;
    private double dealerMarkUp;

    public Vehicle (double cost, double markup, double rate)
    {
      super (rate);
      dealerCost = cost;
      dealerMarkUp = markup;
      listPrice = dealerCost + dealerMarkUp;
    }

    public double getListPrice ()
    {
      return listPrice;
    }
    public double getDealerCost ()
    {
      return dealerCost;
    }
    public double getDealerMarkUp ()
    {
      return dealerMarkUp;
    }
    public void changeMarkUp (double value)
    {
      dealerMarkUp = value;
      listPrice = dealerCost + dealerMarkUp;
    }

}
```

GO ON TO THE NEXT PAGE.

A2 B,

(a) Write the `TaxableItem` method `purchasePrice`. The purchase price of a `TaxableItem` is its list price plus the tax on the item. The tax is computed by multiplying the list price by the tax rate. For example, if the tax rate is 0.10 (representing 10%), the purchase price of an item with a list price of $6.50 would be $7.15.

Complete method `purchasePrice` below.

```
// returns the price of the item including the tax
public double purchasePrice()
{
    return getListPrice() * taxRate;
}
```

Part (b) begins on page 12.

A2 B2

(b) Create the `Vehicle` class, which extends the `TaxableItem` class. A vehicle has two parts to its list price: a dealer cost and dealer markup. The list price of a vehicle is the sum of the dealer cost and the dealer markup.

For example, if a vehicle has a dealer cost of $20,000.00, a dealer markup of $2,500.00, and a tax rate of 0.10, then the list price of the vehicle would be $22,500.00 and the purchase price (including tax) would be $24,750.00. If the dealer markup were changed to $1,000.00, then the list price of the vehicle would be $21,000.00 and the purchase price would be $23,100.00.

Your class should have a constructor that takes dealer cost, the dealer markup, and the tax rate as parameters. Provide any private instance variables needed and implement all necessary methods. Also provide a public method `changeMarkup`, which changes the dealer markup to the value of its parameter.

```
public class vehicle extends TaxableItem {
    private double myCost;
    private double myMarkup;
    private double myRate;

    public Vehicle (double cost, double markup, double rate){
        myCost = cost;
        myMarkup = markup;
        myRate = rate;
    }
    public double getListPrice(){
        return myCost + myMarkup;
    }

    public void changeMarkup (double newMarkup){
        myMarkup = newMarkup;
    }
    public double purchasePrice( ){
        return getListPrice()* myRate;
    }
}
```

GO ON TO THE NEXT PAGE.

A2 C,

(a) Write the `TaxableItem` method `purchasePrice`. The purchase price of a `TaxableItem` is its list price plus the tax on the item. The tax is computed by multiplying the list price by the tax rate. For example, if the tax rate is 0.10 (representing 10%), the purchase price of an item with a list price of $6.50 would be $7.15.

Complete method `purchasePrice` below.

```
// returns the price of the item including the tax
public double purchasePrice()
{
    int p = getListPrice();
    int tp = p + taxRate * p;
    return tp;
}
```

Part (b) begins on page 12.

A2 C2

(b) Create the `Vehicle` class, which extends the `TaxableItem` class. A vehicle has two parts to its list price: a dealer cost and dealer markup. The list price of a vehicle is the sum of the dealer cost and the dealer markup.

For example, if a vehicle has a dealer cost of $20,000.00, a dealer markup of $2,500.00, and a tax rate of 0.10, then the list price of the vehicle would be $22,500.00 and the purchase price (including tax) would be $24,750.00. If the dealer markup were changed to $1,000.00, then the list price of the vehicle would be $21,000.00 and the purchase price would be $23,100.00.

Your class should have a constructor that takes dealer cost, the dealer markup, and the tax rate as parameters. Provide any private instance variables needed and implement all necessary methods. Also provide a public method `changeMarkup`, which changes the dealer markup to the value of its parameter.

```
Public class vehicle extends TaxableItem
  {
    Privak double taxRate;
    Privak double markup;
   Publie abstract double getListPric()
     {
        Super;
     }
   Public vehicle (double rate, double up)
     {
        taxrate = rate;
        markup = up;
     }
    Publie Change Markup (int n)
    {
      markup = n;
    }
   Public double PurchasePrice ()
    {
     int p = getListPrice();
     int mp = p + markup;
     int tp = mp * taxrate;
     return tp;
      }
}
```

AP® COMPUTER SCIENCE A
2006 SCORING COMMENTARY

Question 2

Overview

This question focused on students' ability to design a hierarchy of classes using inheritance. An `Item` interface was provided, along with an abstract `TaxableItem` class that implemented the interface. The `TaxableItem` class contained a private data field for storing a tax rate, a constructor for initializing that field, and an abstract method for accessing its list price. In part (a) students were required to complete the additional `purchasePrice` method, which calculated the purchase price for a `TaxableItem` using its tax rate and list price. In part (b) students were required to design and implement a `Vehicle` class, which was derived from `TaxableItem`. This involved declaring private data fields for storing dealer cost and markup, initializing those fields in a constructor (and using `super` to initialize the tax rate field from `TaxableItem`), implementing the abstract `getListPrice` method, and defining a method for changing the dealer markup.

Sample: A2A
Score: 9

The student correctly answers parts (a) and (b), earning full credit. The implementation of the `getListPrice` method is correct. The student declares a private instance variable to store the list price and correctly initializes `listPrice` in the constructor. The `listPrice` and `dealerMarkUp` fields are updated in the `changeMarkup` method.

Sample: A2B
Score: 6

The student earned a total of 1½ points for part (a): 1 point for correctly calling the `getListPrice` method and a ½ point for returning the calculation in part (a). The calculation, however, is incorrect.

The student earned a total of 4½ points for part (b): a ½ point for the class header, 1 point for correctly declaring the private instance variables, a ½ point for writing a correct constructor header, and 1 point for correctly initializing the private instance variables with the parameters. No credit was earned for calling the parent constructor. The student earned 1 point for correctly implementing the `changeMarkup` method and 1½ points for correctly implementing the `getListPrice` method. However, a 1 point deduction was received for reimplementing the `purchasePrice` method.

Sample: A2C
Score: 3

The student earned a total of 1½ points for part (a): 1 point for correctly calling the `getListPrice` method and a ½ point for returning the calculation in part (a). The calculation is incorrect. The value returned must be a `double` value, not an `int`.

The student earned a total of 1½ points for part (b): a ½ point for the class header and 1 point for correctly declaring the private instance variables. The constructor header is written incorrectly. It must have three `double` parameters (or three parameters whose types match the instance variables) to earn this ½ point. The attempt to initialize the private instance variables is correct and earned a ½ point. The response did not earn the ½ point for correctness since both the dealer cost and the mark up fields must be initialized.

Question 2 (continued)

The `changeMarkup` method header does not have a return type and has the wrong parameter type (`int`). The assignment of the parameter to the markup field in the `changeMarkup` method is correct. The student earned a ½ point for writing the `changeMarkup` method. The implementation of the `getListPrice` method is missing. The student received a 1 point deduction for reimplementing the `purchasePrice` method.

.