Classes and Methods with If Statements

Principles of Computer Programming I

Spring/Fall 20XX

AUGUSTA UNIVERSITY

Outline

- Setters with input validation
  - Properties with input validation
- Constructors with input validation
- Methods with Boolean parameters
- Non-accessor methods using if statements
- Boolean instance variables

Accessors and Encapsulation

- Getter and Setters = accessor methods
- Instance variable is declared `private`, accessor methods are declared `public`
- Ensure that other code can only read (getter) or change (setter) attribute with the object's "permission"

```
class Item
{
  private string description;
  private decimal price;

  public decimal GetPrice()
  {
    return price;
  }
  public void SetPrice(decimal p)
  {
    price = p;
  }
}
```

Like Book or DVD, but more general-purpose

Getter for price attribute

Setter for price attribute

Advantages of Encapsulation

- Accessors can protect instance variables: ensure they are only set to "reasonable" values
- Example: Item's price must be non-negative

```
public void SetPrice(decimal p)
{
  if(p >= 0)
    price = p;
  else
    price = 0;
}
```

Ensure new value is valid before changing attribute

If not, use nearest valid value

- No way to set price to a negative value now!

Setters with Input Validation

- Length and width of a `Rectangle` must not be negative

```
public void SetLength(int newLength)
{
  if(newLength >= 0)
    length = newLength;
  else
    length = 0;
}
```

  - Alternative design: *Ignore* invalid input, leave attribute unchanged

```
public void SetWidth(int newWidth)
{
  if(newWidth >= 0)
    width = newWidth;
}
```

Conditional Operator Validation

- Validation with default: A good time to use conditional operator

In class Item:

```
public void SetPrice(decimal p)
{
  price = (p >= 0) ? p : 0;
}
```

In class Rectangle:

```
public void SetLength(int newLength)
{
  length = (newLength >= 0) ? newLength : 0;
}
```

Review: Properties

- Property: Shortcut for writing a getter and setter
- Use it like a variable instead of a method

```
Item myBook = new Item();
myBook.Price = 14.95m;
myBook.Description = "Bleak House";
Console.WriteLine($"{myBook.Description}" +
    $" costs {myBook.Price:C});
```

Instance variable

```
private decimal price;
public decimal Price
{
  get
  {
    return price;
  }
  set
  {
    price = value;
  }
}
```

Property type

Property name

Properties with Validation

In class Rectangle:

```
public int Length
{
  get
  {
    return length;
  }
  set
  {
    if(value >= 0)
      length = value;
  }
}
```

In class Item:

```
public decimal Price
{
  get
  {
    return price;
  }
  set
  {
    price = (value >= 0) ? value : 0;
  }
}
```

More Advanced Validation

- Recall the Time class:
- Minutes must be between 0 and 59
- Seconds must be between 0 and 59
- `AddMinutes` respects this, but `SetMinutes` does not

```
class Time
{
  private int hours;
  private int minutes;
  private int seconds;
  ...
  public void AddMinutes(int numMinutes)
  {
    int newMinutes = minutes + numMinutes;
    minutes = newMinutes % 60;
    hours += newMinutes / 60;
  }
}
```

More Advanced Validation

- Check if `newMinutes` is within a valid range
  - To compare with a range of values, combine 2 inequality conditions
- What to do if it isn't valid?

```
public void SetMinutes(int newMinutes)
{
  if(newMinutes >= 0 && newMinutes < 60)
  {
    minutes = newMinutes;
  }
  else if(newMinutes >= 60)
  {
    minutes = 59;
  }
  else
  {
    minutes = 0;
  }
}
```

Outline

- Setters with input validation
  - Properties with input validation
- **Constructors with input validation**
- Methods with Boolean parameters
- Non-accessor methods using if statements
- Boolean instance variables

Creating Objects

Instantiation has 3 steps:

1. Create an object
2. Run its constructor
3. Assign the variable a reference to it

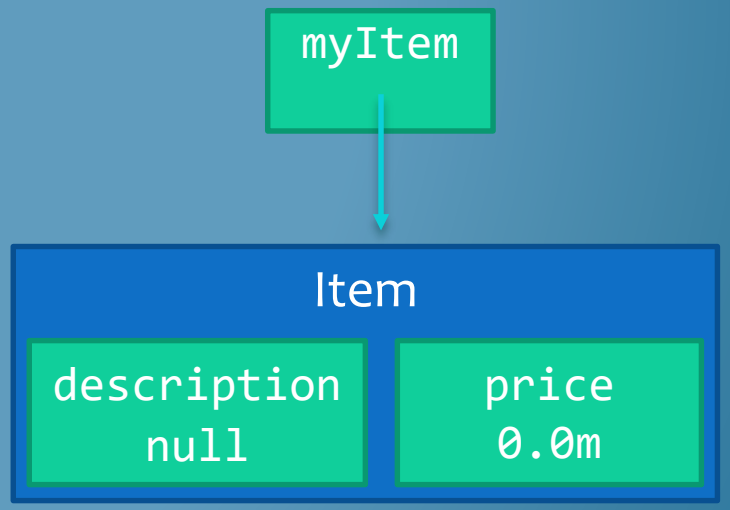
- Constructor's job: Initialize the object
  - Assign values to instance variables
  - Like a "setter" for all of them

```
Item myItem = new Item();
```

Assign reference to myItem

Create object

Run constructor



Constructors With Validation

- Like setters, constructors can validate parameter values before using them – ensure object is not created with "bad" attributes

```
class Item
{
  private string description;
  private decimal price;
  public Item(string initDesc, decimal initPrice)
  {
    description = initDesc;
    price = (initPrice >= 0) ? initPrice : 0;
  }
  ...
}
```

Constructors With Validation

- Room numbers for ClassRoom should be 100-399

```
class ClassRoom
{
  private string building;
  private int number;
  public ClassRoom(string buildingParam, int numberParam)
  {
    building = buildingParam;
    if(numberParam >= 400)
      number = 399;
    else if(numberParam < 100)
      number = 100;
    else
      number = numberParam;
  }
```

Writing a "Smarter" Constructor

- 3-parameter constructor for Time class: hours, minutes, seconds
- If user supplies values ≥ 60, how should we handle it?

```
Time classTime = new Time(0, 75, 0);
```

- Instead of rejecting the value, just "correct" it: 75 minutes is 1 hour 15 minutes

```
Console.WriteLine($"{classTime.GetHours()} hours,"
  + $" {classTime.GetMinutes()} minutes");
```

↓

```
1 hours, 15 minutes
```

Writing a "Smarter" Constructor

```
public Time(int hourParam,
    int minuteParam, int secondParam)
{
  hours = hourParam;
  if(minuteParam >= 60)
  {
    minutes = minuteParam % 60;
    hours += minuteParam / 60;
  }
  else if(minuteParam < 0)
  {
    minutes = 0;
  }
  else
  {
    minutes = minuteParam;
  }
```

```
  if(secondParam >= 60)
  {
    seconds = secondParam % 60;
    minutes += secondParam / 60;
  }
  else if(secondParam < 0)
  {
    seconds = 0;
  }
  else
  {
    seconds = secondParam;
  }
}
```

Problem?

Writing a "Smarter" Constructor

```
public Time(int hourParam,
    int minuteParam, int secondParam)
{
  hours = hourParam;
  if(minuteParam >= 60)
  {
    minutes = minuteParam % 60;
    hours += minuteParam / 60;
  }
  else if(minuteParam < 0)
  {
    minutes = 0;
  }
  else
  {
    minutes = minuteParam;
  }
```

```
  if(secondParam >= 60)
  {
    seconds = secondParam % 60;
    AddMinutes(secondParam / 60);
  }
  else if(secondParam < 0)
  {
    seconds = 0;
  }
  else
  {
    seconds = secondParam;
  }
}
```

Outline

- Setters with input validation
  - Properties with input validation
- Constructors with input validation
- **Methods with Boolean parameters**
- Non-accessor methods using if statements
- Boolean instance variables

Boolean Parameters

- `bool` parameter: Indicate whether the method should take one of 2 different actions
- Example: Room's `ComputeArea` could return either feet or meters, instead of having `ComputeAreaFeet`

```
public double ComputeArea(bool useMeters)
{
  if(useMeters)
    return length * width;
  else
    return GetLengthFeet() * GetWidthFeet();
}
```

Same as body of `ComputeAreaFeet()`

Using Boolean Parameters

```
Console.WriteLine("Compute area in feet (f) or meters (m)?");
char userChoice = char.Parse(Console.ReadLine());
if(userChoice == 'f')
{
  Console.WriteLine($"Area: {myRoom.ComputeArea(false)}");
}
else if(userChoice == 'm')
{
  Console.WriteLine($"Area: {myRoom.ComputeArea(true)}");}
else
{
  Console.WriteLine("Invalid choice");
}
```

Using Boolean Parameters

- Argument can be `true`, `false`, or any Boolean condition

```
Console.WriteLine("Compute area in feet (f) or meters (m)?");
char userChoice = char.Parse(Console.ReadLine());
bool wantsMeters = userChoice == 'm';
Console.WriteLine($"Area: {myRoom.ComputeArea(wantsMeters)}");
```

true if the user entered 'm'

No `if` needed

  - Without a `bool` variable:

```
char userChoice = char.Parse(Console.ReadLine());
Console.WriteLine($"Area: {myRoom.ComputeArea(userChoice == 'm')}");
```

  - Checking for 'f' instead of 'm':

```
char userChoice = char.Parse(Console.ReadLine());
Console.WriteLine($"Area: {myRoom.ComputeArea(userChoice != 'f')}");
```

Room Constructors

- One constructor for meters, one constructor for feet
- "Feet" constructor couldn't initialize the name, otherwise signature would not be unique

Signature: `Room(double, double)`

```
public Room(double lengthFeet, double widthFeet)
{
  length = lengthFeet * 0.3048;
  width = widthFeet * 0.3048;
  name = "Unknown";
}
```

```
public Room(double lengthM, double widthM, string initName)
```

A Flexible Constructor

```
public Room(double lengthP, double widthP, string nameP, bool meters)
{
  if(meters)
  {
    length = lengthP;
    width = widthP;
  }
  else
  {
    length = lengthP * 0.3048;
    width = widthP * 0.3048;
  }
  name = nameP;
}
```

true if the other parameters are in meters, false if they are in feet

Use like this:

```
Room roomFt = new Room(12.5, 10.5, "Bedroom", false);
Room roomM = new Room(6.2, 4.6, "Living Room", true);
```

Outline

- Setters with input validation
  - Properties with input validation
- Constructors with input validation
- Methods with Boolean parameters
- **Non-accessor methods using if statements**
- Boolean instance variables

If Statements with ClassRoom

- In a Main method, with a ClassRoom object named myRoom:

```
if(myRoom.GetNumber() >= 300)
{
  Console.WriteLine("Third floor");
}
else if(myRoom.GetNumber() >= 200)
{
  Console.WriteLine("Second floor");
}
else if(myRoom.GetNumber() >= 100)
{
  Console.WriteLine("First floor");
}
else
{
  Console.WriteLine("Invalid room");
}
```

- Same behavior can be written as a method of ClassRoom

If Statements with `ClassRoom`

- A method that returns a description of which floor the classroom is on:

Inside class `ClassRoom`:

```
public string GetFloor()
{
  if(number >= 300)
    return "Third floor";
  else if(number >= 200)
    return "Second floor";
  else if(number >= 100)
    return "First floor";
  else
    return "Invalid room";
}
```

Instance variable

In the `Main` method:

```
Room myRoom = new Room("UH", 127);
Console.WriteLine(myRoom.GetFloor());
```

A Method for the Prism Class

- Prism has a length, width, and depth:

```
class Prism
{
  private float length;
  private float width;
  private float depth;
  ...
}
```

- Write a method `MakeCube()` that will make all dimensions equal to the *smallest* of the 3
  - Shrink until it is a cube
  - If length = 4.4, width = 3.5, depth = 3.6, then `MakeCube()` should set length and depth to 3.5

A Method for the Prism Class

```
public void MakeCube()
{
  if(length < width && length < depth)
  {
    width = length;
    depth = length;
  }
  else if(width < length && width < depth)
  {
    length = width;
    depth = width;
  }
```

```
  else
  {
    length = depth;
    width = depth;
  }
}
```

Outline

- Setters with input validation
  - Properties with input validation
- Constructors with input validation
- Methods with Boolean parameters
- Non-accessor methods using if statements
- **Boolean instance variables**

Boolean Instance Variables

- `bool` instance variable = an object has an attribute that can only be 1 of 2 states
  - True or false
  - On or off
  - Feet or Meters
- Example: Item is taxable
  - No sales tax on most food
  - Easier to compute final price if item "knows" its tax status

Enhanced Item Class

Item

– description: string
– price: decimal
– taxable: bool
+ SALES_TAX: decimal

Constant: Sales tax percentage

Underline = constant

+ Item(initDesc: string, initPrice: decimal, isTaxable: bool)
+ SetDescription(descParam: string)
+ GetDescription() : string
+ SetPrice(priceParam: decimal)
+ GetPrice() : decimal
+ SetTaxable(taxableParm: bool)
+ IsTaxable() : bool

Convention: Getter is named **IsTaxable**, not GetTaxable

Enhanced Item Constructor

```
class Item
{
  private string description;
  private decimal price;
  private bool taxable
  public const decimal SALES_TAX = 0.08m;
  public Item(string initDesc, decimal initPrice, bool isTaxable)
  {
    description = initDesc;
    price = (initPrice >= 0) ? initPrice : 0;
    taxable = isTaxable;
  }
  ...
```

Displaying Sales Tax

- To display final price of an item, we could compute sales tax in the `Main` method:

```
Item myItem = new Item("Blue Polo Shirt", 19.99m, true);
decimal totalPrice = myItem.GetPrice();
if(myItem.isTaxable())
{
  totalPrice = totalPrice + (totalPrice * Item.SALES_TAX);
}
Console.WriteLine($"Final price: {totalPrice:C}");
```

- Or we could enhance the getter for the price attribute

A Smarter Getter

- `GetPrice()` can automatically include tax if applicable:

```
public decimal GetPrice()
{
  if(taxable)
    return price + (price * SALES_TAX);
  else
    return price;
}
```

Instance variable is unchanged, but return value includes tax

In `Main` method:

```
Item myItem = new Item("Blue Polo Shirt", 19.99m, true);
decimal totalPrice = myItem.GetPrice();
Console.WriteLine($"Final price: {totalPrice:C}");
```