11/9/22

5.4 & 5.5
Accessor Methods and Mutator Methods

Creating Classes (recap)

- **Instance Variables & Methods**
- **Public vs Private**
- **Constructors**
- Accessor Methods
- Mutator Methods

- Instance Variables...

Creating Classes (recap)

- **Instance Variables & Methods**
- **Public vs Private**
- **Constructors**
- Accessor Methods
- Mutator Methods

- Instance Variables (a.k.a attributes, properties, fields) hold the data of an Object

Creating Classes (recap)

- **Instance Variables & Methods**
- **Public vs Private**
- **Constructors**
- Accessor Methods
- Mutator Methods

- Instance Variables (a.k.a attributes, properties, fields) hold the data of an Object
- Methods...

Creating Classes (recap)

- **Instance Variables & Methods**
- **Public vs Private**
- **Constructors**
- Accessor Methods
- Mutator Methods

- Instance Variables (a.k.a attributes, properties, fields) hold the data of an Object
- Methods define the behaviors of an Object

Creating Classes (recap)

- **Instance Variables & Methods**
- **Public vs Private**
- **Constructors**
- Accessor Methods
- Mutator Methods

- Instance Variables (a.k.a attributes, properties, fields) hold the data of an Object
- Methods define the behaviors of an Object
- `public` instance variables...

Creating Classes (recap)

- **Instance Variables & Methods**
- **Public vs Private**
- **Constructors**
- Accessor Methods
- Mutator Methods

- Instance Variables (a.k.a attributes, properties, fields) hold the data of an Object
- Methods define the behaviors of an Object
- `public` instance variables and methods are **accessible from outside** an Object

Creating Classes (recap)

- **Instance Variables & Methods**
- **Public vs Private**
- **Constructors**
- Accessor Methods
- Mutator Methods

- Instance Variables (a.k.a attributes, properties, fields) hold the data of an Object
- Methods define the behaviors of an Object
- `public` instance variables and methods are **accessible from outside** an Object
- `private` instance variables...

Creating Classes (recap)

- **Instance Variables & Methods**
- **Public vs Private**
- **Constructors**
- Accessor Methods
- Mutator Methods

- Instance Variables (a.k.a attributes, properties, fields) hold the data of an Object
- Methods define the behaviors of an Object
- `public` instance variables and methods are **accessible from outside** an Object
- `private` instance variables and methods are **only accessible from within** the Object

Creating Classes (recap)

- **Instance Variables & Methods**
- **Public vs Private**
- **Constructors**
- Accessor Methods
- Mutator Methods

- Instance Variables (a.k.a attributes, properties, fields) hold the data of an Object
- Methods define the behaviors of an Object
- `public` instance variables and methods are **accessible from outside** an Object
- `private` instance variables and methods are **only accessible from within** the Object
- Constructors...

Creating Classes (recap)

- **Instance Variables & Methods**
- **Public vs Private**
- **Constructors**
- Accessor Methods
- Mutator Methods

- Instance Variables (a.k.a attributes, properties, fields) hold the data of an Object
- Methods define the behaviors of an Object
- `public` instance variables and methods are **accessible from outside** an Object
- `private` instance variables and methods are **only accessible from within** the Object
- Constructors define how a Class is initialized when it is created with `new`

Creating Classes

- Variables & Methods
- Public vs Private
- Constructors
- **Accessor Methods**
- Mutator Methods

- `public` methods used to provide read-only access to `private` instance variables within the Object
- Sometimes these are called "get methods" or "getters"
- These `public` methods have a return type that matches the type of the `private` variable being returned

Creating Classes

- Variables & Methods
- Public vs Private
- Constructors
- **Accessor Methods**
- Mutator Methods

- Accessor methods are commonly named `get+VariableName` and do not have parameters

```
public class Person
{
  private String name;

  public Person(String initName) {
    name = initName;
  }

  public String getName() {
    return name;
  }
}
```

Creating Classes

- Variables & Methods
- Public vs Private
- Constructors
- **Accessor Methods**
- Mutator Methods

- Accessor methods can also return a filtered or transformed `private` variable value

```
public class Person
{
  private String phoneNumber;

  public String getAreaCode() {
    return phoneNumber.substring(0,3);
  }
}
```

Principle of Least Privilege

Principle of Least Privilege

- Writing safe and secure code is top of mind for all professional engineers
- The **Principle of Least Privilege** is a pattern that can be used to ensure that you are defensively writing code that is not leaking data (or allowing unintentional access)
- *"Limit access rights to only what is strictly required to perform an expected function."*
  - **Code** that uses a `String` should be able to access methods **about** the `String` but should not be able to inspect or modify any of the `private` internals of the `String` class
  - **Users** should be granted permission to read, write or execute only the files or resources necessary to do their jobs. **Don't give everyone Admin access!**

Wikipedia: [Principle of least privilege]

Principle of Least Privilege

How might you apply the
Principle of Least Privilege when designing a Class?

Principle of Least Privilege

How might you apply the
Principle of Least Privilege when designing a Class?

1. Make all instance variables `private`

Principle of Least Privilege

How might you apply the
Principle of Least Privilege when designing a Class?

1. Make all instance variables `private`

2. If a specific instance variable needs public access - add an Accessor Method

Principle of Least Privilege

How might you apply the Principle of Least Privilege when designing a Class?

1. Make all instance variables `private`

2. If a specific instance variable needs public access - add an Accessor Method

3. If you are VERY confident in how your class will be used (and how it will evolve in the future) you can alternatively elevate the instance access to `public`.

Principle of Least Privilege

How might you apply the Principle of Least Privilege when designing a Class?

1. Make all instance variables `private`

2. If a specific instance variable needs public access - add an Accessor Method

3. If you are VERY confident in how your class will be used (and how it will evolve in the future) you can alternatively elevate the instance access to `public`. ***WARNING: Once you grant this level of access it can be very difficult to take it back!***

Creating Classes

- Variables & Methods
- Public vs Private
- Constructors
- Accessor Methods
- **Mutator Methods**

- `public` methods used to modify internal `private` instance variables
- Sometimes these are called "set methods" or "setters"
- These `public` methods typically have a `void` return type and a parameter that matches the type of the `private` instance variable being modified

Creating Classes

- Variables & Methods
- Public vs Private
- Constructors
- Accessor Methods
- **Mutator Methods**

- Mutator methods are commonly named `set+VariableName` and have a single parameter

```
public class Person
{
  private String name;

  public Person(String initName) {
    name = initName;
  }

  public String getName() {
    return name;
  }

  public void setName(String newName) {
    name = newName;
  }
}
```

Creating Classes

- Variables & Methods
- Public vs Private
- Constructors
- Accessor Methods
- **Mutator Methods**

- Mutator methods can also filter, verify, or transform a value before assigning it to a private instance variable value

```
public class Person
{
  private String areaCode;

  public void setAreaCode(String phoneNumber) {
    areaCode = phoneNumber.substring(0,3);
  }
}
```

Practice on your own

- CSAwesome 5.4 - Accessor Methods
- CSAwesome 5.5 - Mutator Methods
- Accessor and Mutator Methods Exercise in Replit