Behavior, Identity, State

The basics of an object.
Object Behavior

- What does an object do?
- What can you do to an object?
- How does an object respond?

- Just method calls.

- All objects instantiated from the same class will have similar behaviors.
  - Because have same methods.
  - But can be modified by the object’s state and by parameters to the method.
Object Identity

- How can you tell one object from another?
  - especially if they come from the same class and have the same behavior.

- Give objects an identity.

- Identity is the variable name when instantiated.
Object State

Information about the object, stored as instance variables.

- Defining characteristics.
  - e.g., Person class
    - has age, weight, height, eye color, name, etc.
  - e.g., Book class
    - has number of pages, font, the actual text, etc.

- Current state of an object is the current values of all the instance variables.
Encapsulation of State

- In general want to hide an object’s state.
  - Why?
  - So nobody else can abuse it.

  - e.g., AlarmSystem has an entryCode. Don’t want everybody to see it!

  - e.g., Customer has creditCardNumber. They don’t want everybody to see it!

- Hiding state is called encapsulation.
  - keep it in a hidden “capsule”.
How Encapsulate State?

- Hide it by making all the instance variables private.

```java
public class AirlineCustomer {
    private String name;
    private String creditCardNumber;
    private int age;
    private boolean troubleMaker;
    private boolean prefersWindow;
}
```

- Umm, ok. How do I access these variables?
Use Setters (Mutators)

If need to *set* a variable, write a method to do it.

- Called a *setter* method. Or a *mutator* method.
- Convention: method name starts with “set”.
  - public void setAge(int age)
  - public void setName(String name)

```java
public class AirlineCustomer {
    private int age;
    
    public void setAge(int age) {
        this.age = age;
    }
}
```
Use Getters (Accessors)

- If need to get a variable, write a method to get it.
  - Called a *getter* method. Or an *accessor* method.
  - Convention: method name starts with “get”.
    - public int getAge()
    - public String getName()
  - Convention: if boolean instance variable, method name starts with “is”
    - public boolean isTroubleMaker()
    - public boolean isHappy()
  - Sometimes violated: public boolean prefersWindow()
    - just makes more sense this way.
Why Bother With Getters and Setters?

- Can control any access and changes to the state.
  - In getter method can prevent unauthorized individuals from reading credit card numbers.
  - In setter method can prevent someone from maliciously/accidentally changing your name.

- Can “set” instance variables in the constructor, and then forever deny alterations.
  - Prevent anyone from ever changing your name or other attribute.

- See next two examples.
Controlling Access

Watch how we control access!

```java
public class AirlineCustomer {
    private int age;
    private String name;

    //returns age, but won't allow anyone to get the age of a minor.
    public int getAge()
    {
        int age = 0;
        if (age > 18)
        {
            age = this.age;
        }
        return age;
    }

    public void setName(String name)
    {
        if (!name.equals("The Dude"))
        {
            this.name = name;
        }
    }
}
```

Note alphabetized methods and variable names.

Also put all instance variables before methods.
Preventing Access

Set window preference in constructor, then never allow a change!

```java
public class AirlineCustomer {
    private boolean prefersWindow;
    private String name;

    public AirlineCustomer(boolean prefersWindow, String name) {
        this.prefersWindow = prefersWindow;
        this.name = name;
    }

    // can get the name
    public String getName() {
        return name;
    }

    // can get the window preference, but can’t set it!
    public boolean prefersWindow() {
        return prefersWindow;
    }

    // can set the name
    public void setName(String name) {
        if (!name.equals("The Dude")) {
            prefersWindow = name;
            this.name = name;
        }
    }
}
```

Note that I don’t have to say this.name because there is no confusion.