CSE 1223: Introduction to Computer Programming in Java
User-defined Classes
User-defined classes

- The Java language is infinitely extensible through the use of classes
- Every class available in the Java Standard Library was written by a (team of) programmers
  - Every class from Scanner to ArrayList to TreeMap
- We can also write our own classes and use them just like the ones from the Standard Library
  - Ultimately no difference in how these classes are used
User-defined classes

- We often write our own classes when we have some piece of functionality that we can describe as a unit
  - Keep all of our data together
  - Keep methods (behavior) bundled with our data

- A task-driven class
  - Dice used for a game
    - What kind of behavior do we need?
      - Different numbers of sides (6 sided, 8 sided, 20 sided, etc.)
      - Roll the die
      - Read the result
User-defined classes

Let’s think in terms of *encapsulation* and *objects*

- Each Die will be an object with attributes:
  - Number of Sides
  - Current Face Value
User-defined Class Implementation

```java
public class SimpleDie {
    private int numberOfSides;
    private int currentFace;

    ...
}
```

- Class declaration like program declaration
  - Because programs in Java are actually all implemented as classes themselves
Writing Classes

- Instantiating an object
  - We need to call the constructor to construct a new copy of the Die object:
    ```java
    SimpleDie d1 = new SimpleDie(6);
    ```
  - In this case we will create a new die with 6 sides
  - Just like instantiating any other object
    - All objects are equal in Java – whether in the Standard Library or User-defined
public class SimpleDie{
    private int numberOfSides;
    private int currentFace;

    public SimpleDie(int nSides) {
        this.numberOfSides = nSides;
        // Sets the initial value to be the largest value the
        // die can have
        this.currentFace = this.numberOfSides;
    }
}

User-defined Class Implementation

```java
public class SimpleDie{
    private int numberOfSides;
    private int currentFace

    public SimpleDie(int nSides) { ... }
}
```

What are these keywords?
Public vs. Private

- **private** methods and data can only be accessed *by the class that owns them*
  - A private method can only be called from a method inside the class
  - Private data can only be read by methods inside the class

- **public** methods and data can only be accessed *by any object at all*
  - **main** is always a public method
Why restrict access?

- Helps to ensure *encapsulation* of the data
  - Make sure data can only be changed via ways the class programmer has accounted for
  - Enables class programmer to make promises to the programmers using his/her class
  - Enables the programmer to more easily debug his/her class
Accessory and Mutator Methods

- public methods to allow a programmer to manipulate the data fields in an object
  - Also known as “getter” and “setter” methods

- For SimpleDie:

  roll

  getFace
Accessor and Mutator methods

```java
public class SimpleDie{
    private int numberOfSides;
    private int currentFace;

    public void roll() {
        this.currentFace =
            (int)(Math.random() * this.numberOfSides) + 1;
    }

    public int getFace() {
        return this.currentFace;
    }

    ...
}
```
Methods

- Invoking instance methods
  
  ```
  d1.roll();
  
  int value = d1.getFace();
  ```

- Methods always invoked with call:
  
  ```
  object.methodName( ... )
  ```
Static Methods vs. Instance Methods

- Note that the instance methods DO NOT include the `static` keyword
  - Non-static (or `instance methods`) are tied to individual instances
    - Invoked by using the `object name` followed by the method:
      ```java
      String myString = "abc123";
      char myChar = myString.charAt(2);
      ```
  - Static methods (or `class methods`) and variables are items that are tied to the whole class
    - Invoked by using the `class name` followed by the method:
      ```java
      Integer.parseInt("42");
      ```
Classes must be implemented in a file with the same name as the class

- SimpleDie implemented in SimpleDie.java
- Book implemented in Book.java
- etc.
Your Turn

Consider a class designed to model a clock

- State is hour, minute, second and AM or PM
- Behavior is:
  - By default, the clock is set to midnight - 12:00:00AM (constructor)
  - We can set the hour, minute, second and AM or PM (mutators)
  - The clock can tick another second (mutator)
  - We read the current hour, minute, or second and AM or PM (accessors)
  - We display the time as a String in military time (accessor)

- In a small group, write up this class in Java
public class SimpleClock {

    /* private member variables go here */

    public SimpleClock() {
        /* Constructor code goes here */
    }

    public void setHour(int hour) {
        /* Mutator to set the hour goes here */
    }

    /* Other public methods go here */

}
public class SimpleClock {

    /* private member variables go here */
    private int hours;
    private int minutes;
    private int seconds;
    private boolean morning;

    public SimpleClock() {
        this.hours = 12;
        this.minutes = 0;
        this.seconds = 0;
        this.morning = true;
    }

    public void setHour(int hour) {
        this.hours = hour;
    }

    ...
}