Week 7 (Chapter 6) Lab 2

Write a **Methods** class containing various types of methods and do Chapter 6 Practice Program 3 (a test program for the **Pet** class). For extra credit, write a **Parent** class that has a **Person** instance variable named **child** and protect it from change.

1. Do these method creation exercises:
   - Create a class called **Methods** that has a single instance variable `private int x`, then write the following six methods in the class:
     - A *static* method that takes no parameters, returns no value, and prints “Hello, world!”
     - An instance (non-static) method called `print()` that takes no parameters, returns no value, and prints the current value of the `x` instance variable like this: “`x is: x`”
     - A *setter* method for the `x` instance variable, whose parameter name is also `x`; remember the naming convention for *setter* methods and their return type!
       - The *setter* method must guarantee that `x` is never set to a value less than 0
       - If the *setter* argument is negative, it can either set `x` to 0 or leave `x` unchanged
     - A **Methods** constructor that takes one `int` parameter called `x` and calls the *setter* method to set the value of the `x` instance variable
     - A *getter* method for the `x` instance variable; remember the parameter and return types for *getter* methods!
     - A *static void* method called `print()`, whose single parameter is a **Methods** object, that prints the value of its parameter’s `x` instance variable – this is method overloading.
       - Because this method is static, you do NOT invoke it as a member of an object (e.g., `object.print()`). It is not an instance method, and therefore is not connected to an instance. You simply call it by name, with any appropriate parameters
       - This method can use either the object’s `print()` or its *getter* method to do this
   - Test your methods by creating a main method and calling the methods that you have created.
2. Do Chapter 6 Practice Program 3 (modified), create a test program for the Pet class:
   • Use the class Pet in Week 7’s Source Code folder.
   • Write a program to read data for three Pets and display the following information, using the Pet instance methods:
     • The names of the smallest and largest Pets (by weight).
     • The name of the youngest and oldest Pets.
     • The average weight of the three Pets.
     • The average age of the three Pets.
   • Hints: You can keep track of the smallest/largest and youngest/oldest Pets as you are reading them in – once the first one has been initialized you can assume it is the smallest/largest and youngest/oldest to start, and then compare later Pets against those. You can also accumulate their weights and ages as they are being read in, starting from 0 for both. Be sure to calculate the average age as a double, not by using integer division.
3. For extra credit, write a `Parent` class that has a `Person` instance variable named `child` and protect it from change.
   - The `Parent` class has two instance variables, their `name` as a `String` and their `child` as a `Person`, using the `Person` class you created in the **Week 7 Lab 1 exercises, part 1**.
   - Create a `Parent` constructor that takes two parameters, a `String` and a `Person`, and uses them to set the instance variables.
   - Also provide only getter methods for both `name` and `child`.
   - Following the example in the in-class slides, protect the `child Person` object from being changed by a user of the `Parent` class – copy the `Person` object passed to the constructor when you set the `child` instance variable, and copy the `Person` object in `child` when the `getChild` getter method is called to return it.
   - Finally, write a test program that shows that trying to use `Parent` to modify the `child` object does not work.
   - **Hints:** You can start from the original `PetOwner` class in the Sakai Week 7 → Source Code folder and modify it to be the `Parent` class. You can then modify the `Problems` class from that same Source Code folder to use the `Parent` and `Person` classes, and then print information about the `child before and after` the two Problem examples to show that no changes were made in the `child`. 