PAL Worksheet Classes and Objects CSC20

Ref: CSC15 PAL WS Public Fields

Topics: Constructors, Passing objects as parameters,

Remember: Java has a way to bundle data together called a ***class***. A class definition is like a blueprint that defines what to bundle: it doesn't actually bundle the data but does tell you how data will be bundled in the future. This bundling is called *encapsulation*. Here's a class definition that defines what to bundle:

```
public class TeddyBear{
    //private fields
     private String name;
     private int age;
     private double weight;

     //public constructor
     public TeddyBear(String name, int age, double weight){
            this.name=name;
            this.age=age;
            this.weight=weight;
}
}
```

You can "construct" a TeddyBear "object" using this template that actually bundles this data as follows:

TeddyBear *teddy* = new TeddyBear("PapaBear", 12, 302.87)

The left-hand side of this statement declares a variable called *teddy* of type TeddyBear. The right-hand side creates new memory for *teddy* and calls the constructor to assign "PapaBear" to the field *name*, 12 to the field *age* and 302.87 to the field *weigh*t. The *constructor* is called automatically without programmer intervention. The *"this"* keyword is a **reference** to the variable *teddy.* It is possible to have multiple constructors in a class to customize how you would like to initialize the fields. Each TeddyBear object will have its own bundle of data in memory.

Classes provide *abstraction,* so that once they are written, users can create and use objects by interacting with the *instance* methods of the class without worrying about how they are implemented. Notice teddy has *private* fields so they cannot be accessed directly using dot.notation. This is called *information hiding*

To provide access to and to change fields, classes provide *instance* methods called *getters* and *setters*. Typically, a *toString* method is also provided to return a string representation of the object. These methods are called using the dot.notation on the variable name: e.g. *teddy.toString()* or *teddy.setWeight*(500.9). A typical getter and setter will look like:

public void getName(){ return name;} // return name to calling object

public int setAge(int age){ this.age=age;} // change the age of the calling object

Read more in BJP Chapter 8

Ex.1 Extend the TeddyBear class in a file called *TeddyBear.java* to include getters and setters for each field. Write a toString method that returns the name age and weight in this format:

"I am PapaBear, I am 12 years old and I weigh 302.87 pounds".

Ex.2 Create three objects with variable names, *mamaTeddy*, *papaTeddy* and *babyTeddy* with suitable field values in a class called *TeddyBearMain.java*. Display all three objects using the *toString* method. Change ages, weights and display again.

Objects can communicate with each other. For example, one bear can check if he or she is older than the other. We can create an instance method called *iAmOlderThanYou* as follows:

```
public boolean iAmOlderThanYou(TeddyBear other){
        return (age>other.getAge());
}
```

Ex.3. Extend the *TeddyBear* class to with a location field of type *int.* Bears that are within 10 location units are considered part of the same family. Write an instance method *isFamily* that takes in a *TeddyBear* object as a parameter and return a *boolean* value *true* or *false*

Ex4. Write a *Car* class that bundles make, model, year, color (all *String* type) and speed and milesTravelled (both *double*). Create a regular custom constructor that initializes all these fields. Provide *getters* and *setters* for each field as well as a *toString()* method . Create two instance method call *move* and *brake* that input a decimal mile value and increase or decrease the speed by that amount.

Ex5. Write a driver class in a file called *CarMain.java* that creates atleast two cars. Test all the above methods. *Hint*. Make sure *milesTravelled* reflects the correct mileage.