# Module 6 quiz on methods and classes

**LATEST SUBMISSION GRADE**

100%

1. Making instance variables of a class private **1 / 1 point**

- (●) is an application of abstraction and protects against object users gaining direct access to object state.
- ( ) is an application of abstraction and protects the behaviors of an object from being modified by users.
- ( ) is an application of security and protects against hackers modifying and corrupting our code.
- ( ) is a requirement of Java. A class won't compile unless all fields are declared private.

✓ **Correct**

Correct! Recall that abstraction is simply hiding implementation details from the user. Users of the class don't necessarily need to know the name or data type of the instance variables. They ***do*** need to know what data should be supplied when constructing an instance of the object however.

2. A class method such as addMonthlyBonus(double x) that receives an input parameter and uses it to change the instance variable of an object is called a **1 / 1 point**

- (●) mutator method
- ( ) default constructor
- ( ) constructor
- ( ) accessor method

✓ **Correct**

Correct! The key here is ***change***. When the state of an object is changed, it is also considered mutated.

3. Select all of the following method definitions below that employ the correct syntax to define a constructor for the class Vehicle. A *partial* list of the class instance variables includes: **1 / 1 point**

```
1  int year;
2  double odometer;
3  ...
```

- [x]
```
1  public Vehicle(){
2  }
```

**Correct**

This empty parameter list signals that this is the default constructor. We don't have to write any code in a method definition. This constructor simply creates an object with field variables set to the defaults.

- [ ]
```
1  public void Vehicle(double mileage){
2     odometer = mileage;
3  }
4
```

- [ ]
```
1  public makeVehicle(int year){
2     this.year = year;
3  }
```

- [x]
```
1  public Vehicle(double mileage, int year){
2    this(mileage);
3    this.year = year;
4  }
```

**Correct**

This correctly written constructor calls another constructor first, then sets the year field to the value passed by the parameter *year*.

- [ ]
```
1  public Vehicle(double odometer, int year){
2    odometer = odometer;
3    year = year;
4  }
```

4. When a constructor is written for the class *(select all that apply)* **1 / 1 point**

- [ ] the default constructor can not be recreated.
- [ ] no other constructors can be written.
- [x] the default constructor must be recreated if it is to be used by client programs.

> ✓ **Correct**
> If you want to use the default constructor or make available to other users of your class, you must rewrite it in your code.

- [x] the default constructor is no longer available.

> ✓ **Correct**
> Recall that we would now need to recreate the default constructor in order to use it.

5. When calling one constructor from another constructor **1 / 1 point**

- ( ) two objects are created, one being created by each constructor.
- ( ) the class name is used to refer to the *called* constructor.
- ( ) the keyword *this* is used and must be the last line of the constructor code.
- (•) the keyword *this* is used and must be the first line of the constructor code.

> ✓ **Correct**
> Correct! And if you truly understand what constructors are doing when they execute, this is the only answer that makes sense. ***this*** will perform the initial creation of the reference pointer and the memory space for the state of the object. The rest of the code in the calling constructor will then execute.

6. Consider these two methods which are part of the Student class. Assume that age is a private instance variable of the class. **1 / 1 point**

```
1  public void setAge(int years, int months){
2     age = years * 12 + months;
3  }
4
5  public void setAge(int months){
6     age = months;
7  }
```

These methods are an example of *(select all that apply)*

- [ ] accessors.
- [x] overloading.

**Correct**

Correct! Note that both variables have the same name but have different parameter lists. This is how the compiler will know which definition of the module is being called.

- [x] mutators.

**Correct**

Correct! Note that both definitions of setAge make a change to the state of the object that called the method.

- [ ] overriding.