Supplementary Code from Quiz #2

Note: The solution to question #4 will be discussed in another file. To discuss your grade or request a re-mark, please see Lindsey at any of her office hours until and including Friday, Dec. 16. For a complete list of office hours during the exam period, please see the Practice Exam.

```
public interface Appliance {

    public void setWattage(int wattage);

    public int getWattage();

}

public interface Furniture {

    public String getDimensions();
}

public class Refrigerator implements Appliance, Furniture {

    private String dimensions;
    private int wattage = 200;
    public static int yearOfManufacture = 2016;

    public Refrigerator(String dimensions, int wattage) {
        this.dimensions = dimensions;
        this.wattage = wattage;
    }

    public Refrigerator(String dimensions) {
        this.dimensions = dimensions;
    }

    public void setDimensions(String dimensions) {
        this.dimensions = dimensions;
        System.out.println("Refrigerator setter:  dimensions");
    }

    public String getDimensions() {
        System.out.println("Refrigerator getter:  dimensions");
        return dimensions;
    }
```

```
    public void setWattage(int wattage) {
        this.wattage = wattage;
        System.out.println("Refrigerator setter:  wattage");
    }

    public int getWattage() {
        System.out.println("Refrigerator getter:  wattage");
        return wattage;
    }

    public static String whatIsThis() {
        System.out.println("This is a refrigerator");
        return "This is a refrigerator.";
    }

    public static void setYearOfManufacture(int year) {
        yearOfManufacture = year;
        System.out.println("Refrigerator setter:  yearOfManufacture");
    }

}

public class BubbleBrand extends Refrigerator {

    private String dimensions;
    private static int wattage = 100;
    public static int yearOfManufacture = 2017;
    private String colour;

    public BubbleBrand(String dimensions, int wattage, String colour) {
        super(dimensions, wattage);
        this.colour = colour;
    }

    public BubbleBrand(String dimensions, String colour) {
        super(dimensions);
        this.colour = colour;
    }
```

```
    public static String whatIsThis() {
        return "This is a BubbleBrand Refrigerator";
    }

    public void setNewYearOfManufacture(int year) {
        super.yearOfManufacture = year;
        System.out.println("The year is:  " + super.yearOfManufacture);
    }

    public int getWattage() {
        System.out.println("Is BubbleBrand wattage accessible from here?");
        return wattage;
    }

    public void setDimensions(String dimensions) {
        this.dimensions = dimensions;
    }

    public String getDimensions() {
        return dimensions;
    }

    public static void main(String[] args) {
        // Main method goes here.
    }

}
```

CSC207 – Quiz 2 SOLUTIONS

Tuesday 1 November 2016, AM

1. For this page only, please refer to the Supplementary Code booklet and the `main()` method which we will assume is in class `BubbleBrand`: [10 marks]

```
public static void main(String[] args) {
    Refrigerator ref = new BubbleBrand("18 Litres", "purple");
    BubbleBrand bub = new BubbleBrand("17 Litres", 120, "silver");
    System.out.println(ref.getDimensions());
    System.out.println(ref.yearOfManufacture);
    System.out.println(bub.yearOfManufacture);
    System.out.println(ref.whatIsThis());
}
```

(a) Write the resulting output to the screen:

```
null
2016
2017
This is a refrigerator
This is a refrigerator.
```

(b) List all `Refrigerator` methods that were overridden in the `BubbleBrand` class:

`getDimensions(), setDimensions(), getWattage()`

(c) Both of the `Refrigerator` and `BubbleBrand` classes contain a static method called `whatIsThis()`.

(i) If we changed only the `BubbleBrand` version to an instance method, would the code still compile? Yes / [No]

(ii) If we made the `Refrigerator` version an instance method, while keeping the `BubbleBrand` version static, would the code still compile? Yes / [No]

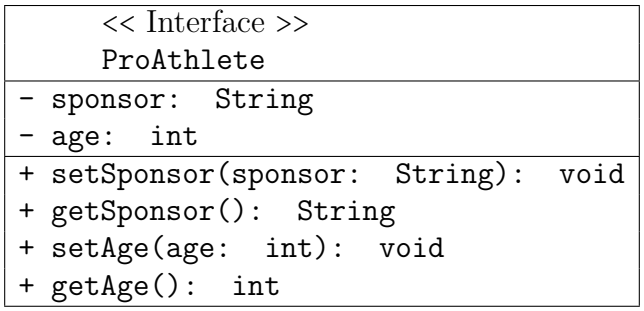
Note: If a parent class and a child class have methods with the same signature except for the word static, this creates a conflict for the compiler: a static method in the parent class should be used because it shadows the method in the child class. But an instance method in the child class should be used because it should overshadow the method in the parent class.

Likewise, the instance method in the parent class should not be used because it should be overshadowed. But the static method in the child class should not be used because it should be shadowed by the method in the parent class.

Both are conflicts that the compiler cannot resolve using the internal logic of the Java language.

2. Consider this UML diagram. [10 marks]

CarDriver
- name: String[] - dob: String - hasLicense: boolean
+ CarDriver(name: String[], dob: String, hasLicense: boolean) + getName(): String[] + setName(name: String[]): void + getDob(): String + setDob(dob: String): void + getHasLicense(): boolean + setHasLicense(hasLicense: boolean): void + toString(): String


<< Interface >>
ProAthlete
- sponsor: String
- age: int
+ setSponsor(sponsor: String): void
+ getSponsor(): String
+ setAge(age: int): void
+ getAge(): int

2. Continued

Implement class `RaceCarDriver` so that it is a subclass of `CarDriver` and also implements `ProAthlete`. (Do **not** draw the UML box. Write the java code.)

```
public class RaceCarDriver extends CarDriver implements ProAthlete {

    private String sponsor;
    private int age;

    public Developer (String name, String dob, boolean hasLicense, String sponsor,
int, age) {
        super(name, dob, hasLicense);
        this.sponsor = sponsor;
        this.age = age;
    }

    public void setSponsor (String, sponsor) {
        this.sponsor = sponsor;
    }

    public String getSponsor() {
        return sponsor;
    }

    public void setAge (int, age) {
        this.age = age;
    }

    public int getAage () {
        return age;
    }

}
```

Alternatively, if you recognized that variables defined in an interface are static and final, and you said that, then you did not have to put any code inside the setter methods because any attempt to set the value of a final variable inside a method will result in a compilation error.

3. Consider this main method: [10 marks]

```
public static void main(String[] args) {

    System.out.println(Triple.getNumberOfTriples()); // prints:   0
    Triple<Integer> tripOne = new Triple<>(new Integer(3), new Integer(4),
            new Integer(5));
    System.out.println(Triple.getNumberOfTriples()); // prints:   1
    Triple<String> tripTwo = new Triple<>("Monday", "Tuesday", "Wednesday");
    System.out.println(Triple.getNumberOfTriples()); // prints:   2
    System.out.println(tripOne.toString()); // prints:   (3, 4, 5)
    System.out.println(tripTwo.toString()); // prints:   (Monday, Tuesday, Wednesday)
    tripTwo.setFirst("hello"); // sets the first of three input values
    System.out.println(tripTwo.getFirst()); // prints:  hello

}
```

Implement class `Triple` so that the main method above compiles, runs, and produces the output shown. You should assume the `main` method is in class `Triple`.

```
public class Triple<T> {
    private T x;
    private T y;
    private T z;
    private static int numberOfTriples = 0;

    public Triple(T x, T y, T z) {
        this.x = x;
        this.y = y;
        this.z = z;
        numberOfTriples++;      }

    public static int getNumberOfTriples() {
        return numberOfTriples;
    }

    public void setFirst(T x) {
        this.x = x;
    }

    public T getFirst() {
        return x;
    }

    public String toString() {
        return "(" + x + ", " + y + "," + z + ")";
```

```
        }
    }
```

CSC207 – Quiz 2 SOLUTIONS

Tuesday 1 November 2016, PM

Student Number: _____

Circle the lecture section in which you are enrolled:

L0101 (WF10)	L0201 (WF12)	L5101 (W6)	L0301 (F2)

---

1. For this page only, please refer to the Supplementary Code booklet and the `main()` method which we will assume is in class `BubbleBrand`: [10 marks]

```
public static void main(String[] args) {
    Refrigerator bub1 = new BubbleBrand("18 Litres", "purple");
    BubbleBrand bub2 = new BubbleBrand("17 Litres", 120, "silver");
    System.out.println(bub1.getWattage());
    bub1.setYearOfManufacture(2018);
    bub2.setYearOfManufacture(2019);
    System.out.println(bub1.whatIsThis());
}
```

(a) Write the resulting output to the screen:

```
Is BubbleBrand wattage accessible from here?
100
Refrigerator setter:  yearOfManufacture
Refrigerator setter:  yearOfManufacture
This is a refrigerator
This is a refrigerator.
```

(b) List all `Refrigerator` methods that were overridden in the `BubbleBrand` class:

`getDimensions(), setDimensions(), getWattage()`

(c) Both of the `Refrigerator` and `BubbleBrand` classes contain a static method called `whatIsThis()`.

(i) If we changed only the `BubbleBrand` version to an instance method, would the code still compile? Yes / [No]

(ii) If we made the `Refrigerator` version an instance method, while keeping the `BubbleBrand` version static, would the code still compile? Yes / [No]

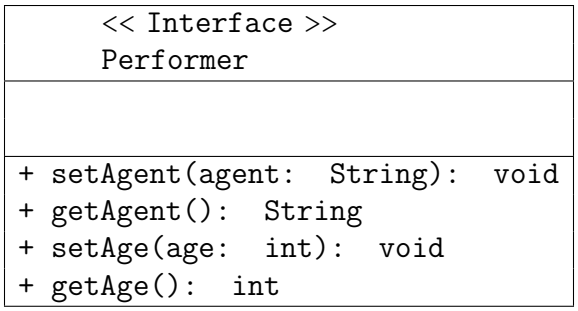
Note: If a parent class and a child class have methods with the same signature except for the word static, this creates a conflict for the compiler: a static method in the parent class should be used because it shadows the method in the child class. But an instance method in the child class should be used because it should overshadow the method in the parent class.

Likewise, the instance method in the parent class should not be used because it should be overshadowed. But the static method in the child class should not be used because it should be shadowed by the method in the parent class.

Both are conflicts that the compiler cannot resolve using the internal logic of the Java language.

2. Consider this UML diagram. [10 marks]

Musician
- name: String[] - dob: String - hasInstrument: boolean
+ Musician(name: String[], dob: String, hasInstrument: boolean) + getName(): String[] + setName(name: String[]): void + getDob(): String + setDob(dob: String): void + getHasInstrument(): boolean + setHasInstrument(hasInstrument: boolean): void + toString(): String


<< Interface >>
Performer
+ setAgent(agent: String): void
+ getAgent(): String
+ setAge(age: int): void
+ getAge(): int

2. Continued

Implement class `OperaSinger` so that it is a subclass of `Musician` and also implements `Performer`. (Do **not** draw the UML box. Write the java code.)

```
  public class OperaSinger extends Musician implements Performer {

    private String agent;
    private int age;

    public Astronaut (String name, String dob, boolean hasInstrument, String agent,
int, age) {
        super(name, dob, hasInstrument);
        this.areaOfSpecialty = agent;
        this.age = age;
    }

    public void setAgent (String, agent) {
        this.agent = agent;
    }

    public String getAgent () {
        return agent;
    }

    public void setAge (int, age) {
        this.age = age;
    }

    public int getAage () {
        return age;
    }

}
```

Alternatively, if you recognized that variables defined in an interface are static and final, and you said that, then you did not have to put any code inside the setter methods because any attempt to set the value of a final variable inside a method will result in a compilation error.

3. Consider this main method: [10 marks]

```
public static void main(String[] args) {

    System.out.println(ThreeTuple.getNumberOfThreeTuples()); // prints:   0
    ThreeTuple<Integer> groupOne = new ThreeTuple<>(new Integer(7), new Integer(8),
            new Integer(9));
    System.out.println(ThreeTuple.getNumberOfThreeTuples()); // prints:   1
    ThreeTuple<String> groupTwo = new ThreeTuple<>("Wednesday", "Thursday",
            "Friday");
    System.out.println(ThreeTuple.getNumberOfThreeTuples()); // prints:   2
    System.out.println(groupOne.toString());
    // prints:   (7, 8, 9)
    System.out.println(groupTwo.toString());
    // prints:   (Wednesday, Thursday, Friday)
    groupTwo.setSecond("hello"); // sets the second of three input values
    System.out.println(groupTwo.getSecond()); // prints:  hello

}
```

Implement class `ThreeTuple` so that the main method above compiles, runs, and produces the output shown. You should assume the `main` method is in class `ThreeTuple`.

```
public class ThreeTuple<T> {
    private T x;
    private T y;
    private T z;
    private static int numberOfThreeTuples = 0;

    public ThreeTuple(T x, T y, T z) {
        this.x = x;
        this.y = y;
        this.z = z;
        numberOfThreeTuples++;      }

    public static int getNumberOfThreeTuples() {
        return numberOfThreeTuples;
    }

    public void setSecond(T y) {
        this.y = y;
    }

    public T getSecond() {
        return y;
```

```
    }

    public String toString() {
        return "(" + x + ", " + y + "," + z + ")";
    }
}
```

CSC207 – Quiz 2 SOLUTIONS

Wednesday 2 November 2016

Student Number: _____

Circle the lecture section in which you are enrolled:

L0101 (WF10)	L0201 (WF12)	L5101 (W6)	L0301 (F2)

---

1. For this page only, please refer to the Supplementary Code booklet and the `main()` method which we will assume is in class `BubbleBrand`: [10 marks]

```
public static void main(String[] args) {
    Refrigerator ref1 = new BubbleBrand("20 Litres", "grey");
    BubbleBrand ref2 = new BubbleBrand("12 Litres", 120, "red");
    System.out.println(ref1.getWattage());
    ref1.setYearOfManufacture(2018);
    System.out.println(ref2.yearOfManufacture);
    System.out.println(ref1.whatIsThis());
}
```

(a) Write the resulting output to the screen:

```
Is BubbleBrand wattage accessible from here?
100
Refrigerator setter:  yearOfManufacture
2017
This is a refrigerator
This is a refrigerator.
```

(b) List all `Refrigerator` methods that were overridden in the `BubbleBrand` class:

`getDimensions(), setDimensions(), getWattage()`

(c) Both of the `Refrigerator` and `BubbleBrand` classes contain a static method called `whatIsThis()`.

(i) If we changed only the `BubbleBrand` version to an instance method, would the code still compile? Yes / [No]

(ii) If we made the `Refrigerator` version an instance method, while keeping the `BubbleBrand` version static, would the code still compile? Yes / [No]

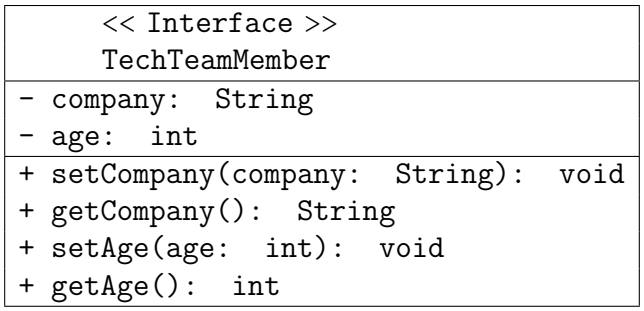
Note: If a parent class and a child class have methods with the same signature except for the word static, this creates a conflict for the compiler: a static method in the parent class should be used because it shadows the method in the child class. But an instance method in the child class should be used because it should overshadow the method in the parent class.

Likewise, the instance method in the parent class should not be used because it should be overshadowed. But the static method in the child class should not be used because it should be shadowed by the method in the parent class.

Both are conflicts that the compiler cannot resolve using the internal logic of the Java language.

2. Consider this UML diagram. [10 marks]

UnionMember
- name: String[] - dob: String - isEmployed: boolean
+ UnionMember(name: String[], dob: String, isEmployed: boolean) + getName(): String[] + setName(name: String[]): void + getDob(): String + setDob(dob: String): void + getIsEmployed(): boolean + setIsEmployed(isEmployed: boolean): void + toString(): String


<< Interface >>
TechTeamMember
- company: String
- age: int
+ setCompany(company: String): void
+ getCompany(): String
+ setAge(age: int): void
+ getAge(): int

2. Continued

Implement class `Developer` so that it is a subclass of `UnionMember` and also implements `TechTeamMember`.

```
public class Developer extends UnionMember implements TechTeamMember {

    private String company;
    private int age;

    public Developer (String name, String dob, boolean isEmployed, String company,
int, age) {
        super(name, dob, isEmployed);
        this.company = company;
        this.age = age;
    }

    public void setCompany (String, company) {
        this.company = company;
    }

    public String getCompany() {
        return company;
    }

    public void setAge (int, age) {
        this.age = age;
    }

    public int getAage () {
        return age;
    }

}
```

Alternatively, if you recognized that variables defined in an interface are static and final, and you said that, then you did not have to put any code inside the setter methods because any attempt to set the value of a final variable inside a method will result in a compilation error.

3. Consider this main method: [10 marks]

```
public static void main(String[] args) {

    System.out.println(TopThree.getNumberOfTopThrees()); // prints:   0
    TopThree<Integer> myTopThree = new TopThree<>(new Integer(2), new Integer(3),
            new Integer(7));
    System.out.println(TopThree.getNumberOfTopThrees()); // prints:   1
    TopThree<String> yourTopThree = new TopThree<>("September", "October", "November");
    System.out.println(TopThree.getNumberOfTopThrees()); // prints:   2
    System.out.println(myTopThree.toString());
    // prints:   (2, 3, 7)
    System.out.println(yourTopThree.toString());
    // prints:  (September, October, November)
    yourTopThree.setFirst("hello"); // sets the first of three input values
    System.out.println(yourTopThree.getFirst()); // prints:   hello
}
```

Implement class `TopThree` so that the main method above compiles, runs, and produces the output shown. You should assume the `main` method is in class `TopThree`.

```
public class TopThree<T> {
    private T x;
    private T y;
    private T z;
    private static int numberOfTopThrees = 0;

    public TopThree(T x, T y, T z) {
        this.x = x;
        this.y = y;
        this.z = z;
        numberOfTopThrees++;      }

    public static int getNumberOfTopThrees() {
        return numberOfTopThrees;
    }

    public void setFirst(T x) {
        this.x = x;
    }

    public T getFirst() {
        return x;
    }

    public String toString() {
```

```
        return "(" + x + ", " + y + "," + z + ")";
    }
}
```

CSC207 – Quiz 2 SOLUTIONS

Friday 4 November 2016

Student Number: _____

Circle the lecture section in which you are enrolled:

L0101 (WF10)	L0201 (WF12)	L5101 (W6)	L0301 (F2)

---

1. For this page only, please refer to the Supplementary Code booklet and the `main()` method which we will assume is in class `BubbleBrand`: [10 marks]

```
public static void main(String[] args) {
    Refrigerator ref1 = new BubbleBrand("20 Litres", "grey");
    BubbleBrand ref2 = new BubbleBrand("12 Litres", 120, "red");
    System.out.println(ref1.getWattage());
    ref1.setYearOfManufacture(2018);
    System.out.println(ref2.yearOfManufacture);
    System.out.println(ref1.whatIsThis());
}
```

(a) Write the resulting output to the screen:

```
Is BubbleBrand wattage accessible from here?
100
Refrigerator setter:  yearOfManufacture
2017
This is a refrigerator
This is a refrigerator.
```

(b) List all `Refrigerator` methods that were overridden in the `BubbleBrand` class:

`getDimensions(), setDimensions(), getWattage()`

(c) Both of the `Refrigerator` and `BubbleBrand` classes contain a static method called `whatIsThis()`.

(i) If we changed only the `BubbleBrand` version to an instance method, would the code still compile? Yes / [No]

(ii) If we made the `Refrigerator` version an instance method, while keeping the `BubbleBrand` version static, would the code still compile? Yes / [No]

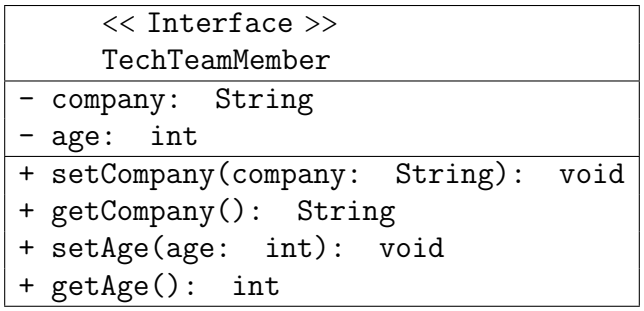
Note: If a parent class and a child class have methods with the same signature except for the word static, this creates a conflict for the compiler: a static method in the parent class should be used because it shadows the method in the child class. But an instance method in the child class should be used because it should overshadow the method in the parent class.

Likewise, the instance method in the parent class should not be used because it should be overshadowed. But the static method in the child class should not be used because it should be shadowed by the method in the parent class.

Both are conflicts that the compiler cannot resolve using the internal logic of the Java language.

2. Consider this UML diagram. [10 marks]

UnionMember
- name: String[] - dob: String - isEmployed: boolean
+ UnionMember(name: String[], dob: String, isEmployed: boolean) + getName(): String[] + setName(name: String[]): void + getDob(): String + setDob(dob: String): void + getIsEmployed(): boolean + setIsEmployed(isEmployed: boolean): void + toString(): String


<< Interface >>
TechTeamMember
- company: String
- age: int
+ setCompany(company: String): void
+ getCompany(): String
+ setAge(age: int): void
+ getAge(): int

2. Continued

Implement class `Developer` so that it is a subclass of `UnionMember` and also implements `TechTeamMember`.
2. Continued

Implement class `Developer` so that it is a subclass of `UnionMember` and also implements `TechTeamMember`.

```
public class Developer extends UnionMember implements TechTeamMember {

    private String company;
    private int age;

    public Developer (String name, String dob, boolean isEmployed, String company,
int, age) {
        super(name, dob, isEmployed);
        this.company = company;
        this.age = age;
    }

    public void setCompany (String, company) {
        this.company = company;
    }

    public String getCompany() {
        return company;
    }

    public void setAge (int, age) {
        this.age = age;
    }

    public int getAage () {
        return age;
    }

}
```

Alternatively, if you recognized that variables defined in an interface are static and final, and you said that, then you did not have to put any code inside the setter methods because any attempt to set the value of a final variable inside a method will result in a compilation error.

3. Consider this main method: [10 marks]

```
public static void main(String[] args) {

    System.out.println(FavThree.getNumberOfFavThrees()); // prints:   0
    FavThree<Integer> myFav = new FavThree<>(new Integer(10), new Integer(20),
            new Integer(30));
    System.out.println(FavThree.getNumberOfFavThrees()); // prints:   1
    FavThree<String> yourFav = new FavThree<>("Spring", "Summer", "Fall");
    System.out.println(FavThree.getNumberOfFavThrees()); // prints:   2
    System.out.println(myFav.toString());
    // prints:   (10, 20, 30)
    System.out.println(yourFav.toString());
    // prints:  (Spring, Summer, Fall)
    yourFav.setFirst("hello"); // sets the first of three input values
    System.out.println(yourFav.getFirst()); // prints:   hello
}
```

Implement class `FavThree` so that the main method above compiles, runs, and produces the output shown. You should assume the `main` method is in class `FavThree`.

```
public class FavThree<T> {
    private T x;
    private T y;
    private T z;
    private static int numberOfFavThrees = 0;

    public FavThree(T x, T y, T z) {
        this.x = x;
        this.y = y;
        this.z = z;
        numberOfFavThrees++;      }

    public static int getNumberOfFavThrees() {
        return numberOfFavThrees;
    }

    public void setFirst(T x) {
        this.x = x;
    }

    public T getFirst() {
        return x;
    }
```

```
    public String toString() {
        return "(" + x + ", " + y + "," + z + ")";
    }
}
```