COMI 2510 Advanced Programming and Design
Assignment 2

Given the attached class named `Person` with fields for holding a person's name, address, and telephone number, extend `Person` with a class named `Customer`. The `Customer` class should have a field for a customer number and a `boolean` field indicated whether the customer wishes to be on a mailing list. Write a no-arg constructor, a copy constructor, and a constructor that takes data to instantiate all fields and write the appropriate mutator and accessor methods for the class's fields. Write a `toString()` method for `Customer`.

Further extend `Customer` by writing a `PreferredCustomer` class. In the preferred customer plan, a customer can earn a discount on all purchases. The amount of a customer's discount is determined by the amount of the customer's cumulative purchases in the store as follows:

- When a preferred customer spends $500, he or she gets a 5% discount on all future purchases.
- When a preferred customer spends $1,000, he or she gets a 6% discount on all future purchases.
- When a preferred customer spends $1,500, he or she gets a 7% discount on all future purchases.
- When a preferred customer spends $2,000 or more, he or she gets a 10% discount on all future purchases.

The `PreferredCustomer` class should have a field for the amount of the customer's purchases. (And no other additional fields.) It should have three constructors: no-arg, copy, and one that takes data to instantiate all fields. It should have a `toString()` method. It should have a method to access the field storing the amount of the customer's purchases, a method to add a purchase amount to that field, and a method `getDiscount()` that returns the amount of the `PreferredCustomer's` discount (i.e. .05, .06, etc.). It should have no other additional methods.

Write a program that tests your `PreferredCustomer` class by creating a `PreferredCustomer` object (using either appropriate constructor) and then repeatedly adding to the stored purchase amount, accessing the discount, and displaying both the object (using its `toString` method) and the discount to the console. Please hard-code this and do not require the user to interact with the class to test it. For example, you might create a `PreferredCustomer` object and then repeatedly make purchases of $250 until the customer has the maximum discount.

Hand in all classes and a class diagram showing all three classes in the inheritance hierarchy.

```
public class Person {

	//Fields:
	private String name;		//the person's name
	private String address;		//the person's address
	private String telephone; //the person's telephone number

	//Methods:
	//CONSTRUCTORS
	/**
	 * Person, no-arg constructor
	 * Initialize all fields to the empty string
	 */
	public Person() {
		name="";
		address="";
		telephone="";
	}

	/**
	 * Person, constructor that takes arguments for all fields
	 * @param name -- String, value to put in the name field
	 * @param address -- String, value to put in the address field
	 * @param telephone -- String, value to put in the telephone field
	 */
	public Person(String name, String address, String telephone){
		this.name=name;
		this.address=address;
		this.telephone=telephone;
	}

	/**
	 * Person, copy constructor
	 * @param toClone -- a Person object to copy into the current object
	 */
	public Person(Person toClone){
		this.name=toClone.name;
		this.address=toClone.address;
		this.telephone=toClone.telephone;
	}

	//ACCESSORS

	/**
	 * getName, name field accessor
	 * @return value in the name field, String
	 */
	public String getName(){
		return name;
	}

	/**
	 * getAddress, address field accessor
	 * @return value in the address field, String
	 */
	public String getAddress(){
		return address;
	}

	/**
	 * getTelephone, telephone field accessor
```

```
	 * @return value in the telephone field, String
	 */
	public String getTelephone(){
		return telephone;
	}

	//MUTATORS

	/**
	 * setName, name field mutator
	 * @param name -- String, value to put in the name field
	 */
	public void setName(String name){
		this.name=name;
	}

	/**
	 * setAddress, address field mutator
	 * @param address -- String, value to put in the address field
	 */
	public void setAddress(String address){
		this.address=address;
	}

	/**
	 * setTelephone, telephone field mutator
	 * @param telephone -- String, value to put in the telephone field
	 */
	public void setTelephone(String telephone){
		this.telephone=telephone;
	}

	//OTHER METHODS

	/**
	 * toString, return a String representing the fields of the object.
	 */
	public String toString(){
		String str="Name: "+name+"\nAddress: "+address+"\nTelephone number:
"+telephone;
		return str;
	}
}
```

Assignment 2 Rubric

Component		Quality			
		Exceptional	**Acceptable**	**Amateur**	**Unsatisfactory**
	Run-time specifications 50%	50 pts: The program meets all of the run-time specifications, with no additional unspecified functionality.*	40 pts: There is additional unspecified functionality or the program produces incorrect results in no more than 5% of the customer's tests.	25 pts: The program produces incorrect results in no more than 10% of the customer's tests.	10 pts: The program produces incorrect results in more than 10% of the customer's tests.
	Design specifications 25%	25 pts: The program meets design specifications. No fields or methods are added or removed. Hierarchy is correct.	20 pts: Classes mostly meet specifications but are off by less than 10% (e.g. additional or missing field or method, incorrect type field or method).	12 pts: Classes mostly meet specifications but are off by less than 20%.	5 pts: Classes do not meet specifications more than 20% of the time.
	Diagramming techniques 12.5%	12.5 pts: Diagramming techniques are used appropriately and accurately reflect the static state of the program.	10 pts: Diagramming techniques are used appropriately but with minor errors that don't affect readability and accurately reflect the static state of the program or have minor errors that don't affect comprehension of the system state.	6.25 pts: Diagramming techniques are used appropriately but with several errors that somewhat affect readability or have minor errors that affect comprehension of the system state in less than 20% of the execution portrayed.	2.5 pts: Diagramming techniques have major errors that affect readability or do not accurately reflect the static state of the program that affect comprehension of the system state. (e.g. wrong member names, types, visibility indicators in class diagram, relationships between classes shown incorrectly)
	Documentation 12.5%	12.5 pts: The program contains comments including the programmer's name and date. Javadoc comments are included as shown in the text for all classes. There are block comments (as many as necessary) for each distinct block of code which accurately describe what the block is accomplishing.	10 pts: The header comment is incomplete but contains name and date, and/or the block comment(s) aren't clear. Javadoc comments are missing components less than 10% of the time.	6.25 pts: The documentation partially meets the exceptional guidelines with several poorly written comments and/or missing comments or missing components less than 25% of the time.	2.5 pts: 25% or more of the comments are missing or unhelpful.

*If you want to change the functional specifications of the program in any way, you must clear it with your customer (the instructor) in writing prior to making the changes. Include documentation of the specification changes when you submit your program.