Problem Solving Session

- The remainder of today's class will comprise the ***problem solving session*** (***PSS***).
- Your instructor will divide you into ***teams of 3 or 4 students***.
- Each team will ***work together*** to solve the following problems over the course of ***20-30 minutes***.
  - You may work on paper, a white board, or digitally as determined by your instructor.
  - You will submit your solution by pushing it to GitHub before the end of class.
- Your instructor will go over the solution before the end of class.
- If there is any time remaining, you will begin work on your homework assignment.

Class participation is a significant part of your grade (20%). This includes in class activities and the problem solving session.

Your Course Assistants will grade your participation by verifying that you pushed your solutions before the end of the class period each day.

Problem Solving Team Members

If you are working digitally, record the name of each of your problem solving team members here.

Do not forget to ***add every team member's name***! Your instructor (or course assistant) may or may not use this to determine whether or not you participated in the problem solving session.

Dessa S.
Eric m.
Uyen Nhi Q.

Problem Solving 1

- A ***product*** has a *name*, a unique 7-digit *product code*, and a *manufacturer's suggested retail price* (MSRP).
- A ***truck*** has a *capacity* that determines how many products can be *loaded* onto the truck. As long as a truck is not *full*, more products can be loaded. A truck can be unloaded one product at a time until it is *empty*.
- A ***factory*** is a place that can *manufacture* a specific kind of product. *All* factories can manufacture, but each one creates different kinds of products.
- A ***complex*** is a collection of *factories* that are geographically close to each other. When trucks arrive, the are *loaded* with a random assortment of products from the factories in the complex.
- A ***microwave*** oven is a product that can be used to cook food. Every microwave has a *size* (in cubic feet), and a *maximum wattage*.
- A ***microwave factory*** is a factory that can manufacture microwaves of different sizes and wattages.

Read the brief descriptions to the left and use the table below to indicate whether you think the final implementation will be an *enum*, *class*, *abstract* class, or *interface*. If you think the type is a subclass of some other type, indicate its super class as well (or type *NA* if it has no parent).

Name	Type	Super Class/ Interface?
Product	Class or interface	N/A
Truck	Class	NA
Factory	Interface :o	N/A
Complex	Class	N/A
Microwave	Class	Product
Microwave Factory	Class	Factory

Type	State	Behavior
Product	Name ID MSRP	<none>
Truck	Capacity products	Load Unload isFull getCapacity
Factory	<none> // Just need to define the manufacturing's functionality	Manufacture product
Microwave Factory	<None>	Manufacture product

Problem Solving 2

Consider only the types listed in the table on the left. Use the problem statement from the previous problem to identify the state and behavior in each class. Underline behavior that you think may be abstract.

(reduce the font size if you need to)

Problem Solving 3

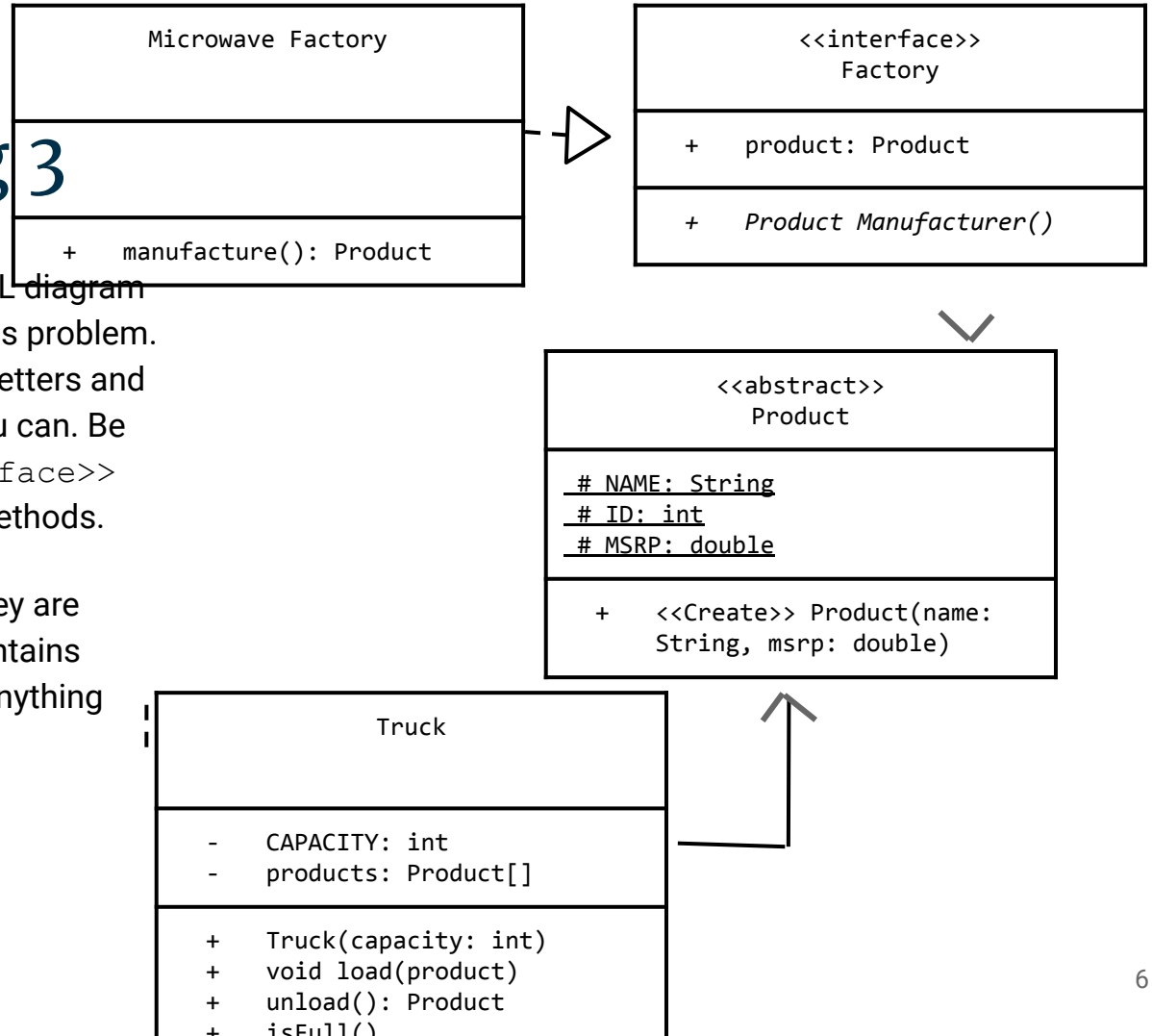
Using the shapes to the right, draw a UML diagram from your domain analysis in the previous problem. You may leave out trivial methods (like getters and setters). Otherwise, be as detailed as you can. Be sure to use `<<abstract>>` or `<<interface>>` labels, and use *italics* for any abstract methods.

Remember: if one class ***uses*** another, they are connected by a ***solid line*** if one class contains another as a field, and a ***dotted line*** for anything else (parameters, return values, etc.).

Problem Solving 3

Using the shapes to the right, draw a UML diagram from your domain analysis in the previous problem. You may leave out trivial methods (like getters and setters). Otherwise, be as detailed as you can. Be sure to use `<<abstract>>` or `<<interface>>` labels, and use *italics* for any abstract methods.

Remember: if one class ***uses*** another, they are connected by a ***solid line*** if one class contains another as a field, and a ***dotted line*** for anything else (parameters, return values, etc.).



```
public class Truck(){
	private static final int MAX_CAPACITY;
	private static int currentCapacity;
	private static Product[] products;

	public Truck(int capacity) {
		this.MAX_CAPACITY = capacity;
		this.currentCapacity = 0;
		This.products = new Product[capacity];
	}
	public void load(Product product){
		If (this.currentCapacity >= MAX_CAPACITY)
			{System.out.printLn(“Truck is full, can
		not load.”)
			 } else {
				this.Product[currentCapacity] =
		product
			this.currentCapacity++
			}
	}
	public Product unload(){
		Product product = Product[currentCapacity -
1];
		this.Product[this.currentCapacity - 1] = null;
		this.currentCapacity--;
		return product;
	}
}
```

Problem Solving 4

Use the space to the left to write the code for the `Truck`. Use your UML diagram from the previous exercise as a guide; focus on the methods to ***load*** and ***unload*** a truck. If you have time, fill in more details.

If you need to create an object, you may *hard code* any constructor parameters for now.

Assume that any other classes that you need already exist.