# Exercises: OOP in TypeScript

## Create a Person Class

Create a class **Person** with the following properties:

* **firstName** (string)
* **lastName** (string)
* **age** (number)

Add a method **introduce()** that returns a string:  
"***My name is <firstName> <lastName> and I am <age> years old.***"

### Example

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| const person = new Person("John", "Doe", 30);  console.log(person.introduce()); | My name is John Doe and I am 30 years old. |

## Bank Account

Create a class **BankAccount** with a private property balance (number).  
Add methods:

* **deposit**(amount: number): void
* **withdraw**(amount: number): void
* **getBalance**(): number

Ensure that withdrawing more than the balance is not allowed.

### Example

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| const account = new BankAccount(100);  account.deposit(50);  account.withdraw(30);  console.log(account.getBalance()); | 120 |
| const account = new BankAccount(20);  account.withdraw(30);  console.log(account.getBalance()); | 20 |

## Simple Inheritance

Create a base class **Vehicle** with:

* property: **brand** (string)
* method: **drive**(): string returning **"Driving a <brand>"**

Create a subclass **Car** that adds:

* property: **model** (string)
* method override: **drive**() returning **"Driving a <brand> <model>"**

### Example

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| const car = new Car("Toyota", "Corolla");  console.log(car.drive()); | Driving a Toyota Corolla |

## Abstraction

Define an interface **Animal** with a method **makeSound**(): string  
Create a class **Dog** that implements this interface.  
The **makeSound** method should return "Woof".

### Example

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| const dog = new Dog();  console.log(dog.makeSound()); | Woof |

## Message Override

Create a base class **Message** with method **send**(): string that returns **"Sending message..."**  
Create two subclasses **EmailMessage** and **SMSMessage**.  
Override the method in each subclass to return:

* "**Sending email...**"
* "**Sending SMS...**" respectively.

### Example

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| const messages: Message[] = [new Message(), new EmailMessage(), new SMSMessage()];  messages.forEach((msg) => {  console.log(msg.send());  }); | Sending message...  Sending email...  Sending SMS... |

## Access Modifiers

Create a class **Employee** with:

* public property **name**
* protected property **position**
* private property **salary**

Implement methods:

* **getDetails** – to return information about name and position (see format output)
* **showSalary** – to return information about salary (see format in output)

### Examples

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| const emp = new Employee("Alice", "Manager", 5000);  console.log(emp.getDetails());  console.log(emp.showSalary());  console.log(emp.name)  console.log(emp.salary)  console.log(emp.position) | Name: Alice, Position: Manager  Salary: $5000  Alice  //Error: Property 'salary' is private  //Error: Property 'position' is protected |

## Class counter

Create a class **Counter** with:

* static property **count** initialized to 0 – property should only be accessible from within the class
* static method **increment**() that increases the count
* static method **getCount**() that returns the current count

### Example

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| Counter.increment();  Counter.increment();  console.log(Counter.getCount()); | 2 |
| Counter.counter = 10; | //Error: only accessible within Counter |

## Readonly Modifier

Create a class **Book** with:

* readonly property **title**
* readonly property **author**

Once set through the **constructor**, these values should not be changed.

### Examples

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| const book = new Book("1984", "George Orwell");  console.log(`${book.title} by ${book.author}`);  book.title = "Brave New World";  book.author = "Terry Pratchet"; | 1984 by George Orwell  // Error: Cannot assign to 'title' because it is a read-only property  // Error: Cannot assign to author because it is a read-only property |

## Abstract Class

Create an abstract class **Shape** with:

* property **color**: string
* abstract method **getArea**(): number

Create two subclasses that implement **getArea**.

* **Circle** with property
  + **radius: number**
* **Rectangle** with properties:
  + **sideA: number**
  + **sideB: number**

### Examples

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| const circle = new Circle("red", 5);  console.log(circle.getArea()); | 78.53981633974483 |
| const rectangle = new Rectangle("blue", 4, 6);  console.log(rectangle.getArea()); | 24 |

## Getters and Setters (Accessors)

Create a class **User** with:

* private **\_username**: string
* **getter** **username**()
* **setter** **username**(newUsername: string) that ensures username is at least 3 characters long, otherwise throws an error
* ensure the user cannot be instantiated without a **valid username**

### Examples

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| const user = new User("Martin");  user.username = "johnDoe";  console.log(user.username); | johnDoe |
| const user = new User("jo"); | Error: Username must be at least 3 characters long |
| const user = new User("Martin");  user.username = "Do"; | Error: Username must be at least 3 characters long |

# 11. Online Course Management System

Create an abstract class **Course** with:

* **title**: string (public)
* **duration**: number in hours (public)
* abstract method **getDescription**(): string

Create two subclasses:

* **ProgrammingCourse** with additional property - **language**: string
* **DesignCourse** with additional property - **tools**: string[]

Each subclass should override **getDescription**() to return a string like:  
"**Programming Course: <title> in <language> - <duration> hours**"  
"**Design Course: <title> using <tools> - <duration> hours**"

### Example

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| const jsCourse = new ProgrammingCourse("Intro to JavaScript", 40, "JavaScript");  const uiCourse = new DesignCourse("UX Fundamentals", 30, ["Figma", "Sketch"]);  console.log(jsCourse.getDescription());  console.log(uiCourse.getDescription()); | Programming Course: Intro to JavaScript in JavaScript - 40 hours  Design Course: UX Fundamentals using Figma, Sketch - 30 hours |

# 12. Simple Task Tracker with Access Control

Create a class **Task** with:

* **title**: string
* **description**: string
* **completed**: boolean (default: false)
* private property **createdBy**: string

Use getters and setters:

* Get **createdBy** only with a getter
* Add method **toggleStatus**(): void to switch completed status
* Add method **getDetails**(): string to return information about a task in the format **"Task: <title> - <description> - <Completed | Pending>"**

Also create a static method **createSampleTasks**(): Task[]  
that returns an array with at least two example tasks.

### Examples

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| const task1 = new Task("Complete homework", "Finish math exercises", "Charlie");  task1.toggleStatus();  console.log(task1.getDetails()); | Task: Complete homework - Finish math exercises - Completed |
| const task2 = new Task("Clean room", "Clean the room", "Mary");  console.log(task2.getDetails()); | Task: Clean room - Clean the room - Pending |
| const tasks = Task.createSampleTasks();  tasks.forEach(task => console.log(task.getDetails())); |  |

# 13. Inventory System

Create a class **Product** with:

* static property **productCount** to track how many products were created, protect it from being changed outside the class
* readonly **id**: number (auto-incremented using **productCount**)
* **name**: string – ensure a products name is at least 1 characters long
* **price**: number – ensure the price is > 0
* **getDetails**(): string method to print info about the product in format **"ID: <id>, Name: <name>, Price: $<price>"**

Create another class **Inventory** with:

* private **products**: Product[]
* method **addProduct**(product: Product): void
* method **listProducts**(): string[] - returns formatted info about product in the inventory each on a new line and then on a new line prints the number of total products in the format **"Total products created: <products count>"**

Use **readonly**, **private**, **static**, **getters** and **setters** and **encapsulation** appropriately.

### Example

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| const inventory = new Inventory();  const product1 = new Product("Laptop", 1200);  const product2 = new Product("Phone", 800);  inventory.addProduct(product1);  inventory.addProduct(product2);  console.log(inventory.listProducts()); | ID: 1, Name: Laptop, Price: $1200  ID: 2, Name: Phone, Price: $800  Total products created: 2 |
| Product.productCount = 10;  const product = new Product("", 800);  const product2 = new Product("Phone", 0);  product.id = 5; | //TS Error: cannot assign to 'productCount'  //Runtime Error: Name must contain at least 1 character  //Runtime Error: Price must be positive  //TS Error: cannot assign to 'id' |

# 14. Overloaded Calculator

Create a class **Calculator** with a single method **calculate**:

* the method should take an **operation** parameter with possible values **'power' | 'log' | 'add' | 'subtract' | 'multiply' | 'divide'**
* and **up to 4 additional number parameters** with which to perform the operation, the number of parameters is based on the operation
  + if the operation is **power** or **log –** allow **exactly** **2** number parameters
  + if the operation is **add**, **subtract**, **multiply** or **divide** allow **between 2 and 4** number parameters
* The operations themselves should function as expected:
  + **power** a b => **ab**
  + **log** a b => **logba**
  + **add** a b c d => **a + b + c + d**
  + **subtract** a b c d => **a - b - c - d**
  + **multiply** a b c d => **a \* b \* c \* d**
  + **divide** a b c d => **a / b / c / d**
* **Hint**: You should use method overloading

### Example

|  |  |
| --- | --- |
| **Sample code** | **Output** |
| const calc = new Calculator();  console.log(calc.calculate('power', 2, 3));  console.log(calc.calculate('power', 4, 1/2));  console.log(calc.calculate('log', 8, 2));  console.log(calc.calculate('add', 10, 5));  console.log(calc.calculate('add', 10, 5, 3));  console.log(calc.calculate('subtract', 10, 5));  console.log(calc.calculate('multiply', 2, 3, 4));  console.log(calc.calculate('divide', 100, 5, 2, 2)); | 8  2  3  15  18  5  24  5 |
| const calc = new Calculator();  console.log(calc.calculate('power', 2, 3, 2));  console.log(calc.calculate('add', 2));  console.log(calc.calculate('log', 2, 3, 4, 5));  console.log(calc.calculate('multiply', 2, 3, 4, 5, 6)); | // TS Error - 'power' not assignable to 'add' | 'subtract' | 'multiply' | 'divide'  // TS Error – Expected 3-5 arguments but got 2  // TS Error - 'log' not assignable to 'add' | 'subtract' | 'multiply' | 'divide'  // TS Error – Expected 3-5 arguments but got 6 |