Python OOP Retake Exam - 11 April 2025  
Tech Service Management



*Welcome to the Tech Service Management!*

*The system consists of different device types (Laptops, Smartphones, Smartwatches) and repair shops that handle maintenance. A central TechServiceManager will coordinate device registrations, repairs, and status tracking.*

***Note: You cannot change the folder and file structure and their names!***

A screenshot of a computer

AI-generated content may be incorrect.

**Judge Upload**

For the **first two problems**, create a **zip** file with the **project** folder and **upload it** to the judge system.

For the **last problem**, create a **zip** file with the **test** folder and **upload it** to the judge system.

You do not need to include your **venv**, **.idea**, **pycache**, and **\_\_MACOSX** (for Mac users) **in** the **zip file**, so you do not exceed **the maximum allowed size** of **16.00 KB**.

## Structure (Problem 1) and Functionality (Problem 2)

Your task is to implement all the **classes' structure and functionality** (properties, methods, inheritance, abstraction, etc.)

You are **free to add additional attributes** (instance attributes, class attributes, methods, dunder methods, etc.) to simplify your code and increase readability if it does not change the project's final result under its requirements so that the program works properly.

### Class BaseDevice

In the **base\_device.py** file, the class **BaseDevice** should be implemented. It is a **base class** for **all types** of electronic **devices**,and it **should not be able to be instantiated**.

#### Structure

The class should have the following attributes:

* **serial\_number:** **str**
  + Represents the **serial number** of the **device**.
  + The serial number **must** contain **only letters and digits**, if not **raise** a ValueError with the message: **"Invalid serial number!"**
* **durability:** **int**
  + The value represents the **durability** of the **device** in percentages.
  + The durability **must** be **between 1** and **100**, **both inclusive**. If not, **raise** ValueError with the message: **"Durability is out of range!"**
* **is\_functional:** **bool**
  + Indicates whether the **device** is **working**.
* **device\_type:** **str**
  + The value represents the **type** of the **device**.
  + The type **cannot** bean **empty string** orcontainonly **white spaces**, if so **raise** a ValueError with the message: **"Type cannot be empty!"**

#### Methods

##### \_\_init\_\_(serial\_number: str, durability: int, is\_functional: bool, device\_type: str)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

##### check\_functionality()

* A **method** that **evaluates** the **functionality status** of the **device**:
  + If the attribute **is\_functional** is already set to **False**, the method **makes no changes**.
  + If **durability** is **less than 2%**, but the device has **not** been **indicated** as **malfunctioning**, set the **is\_functional** attribute to **False**.

##### repair()

* A **method** that **restores durability** to a **certain percentage** and **sets** the new **functional status**.
* Remember that **each** **device** type **implements** the **method differently**.

### Class Laptop

In the **laptop.py** file, the class **Laptop** should be implemented. A laptop is a **type** of **base device**,and its **device\_type** property is **set** to **"Laptop"**.

#### Methods

##### \_\_init\_\_(serial\_number: str, durability: int, is\_functional: bool)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

##### repair()

* If the device **needs repair** (is\_functional=False), the method **increases** its **durability** by **adding 50%** to its **current value**. However, it should **not exceed** the maximum level of **100%**.

Example: If the current durability is 8%, the new value after repair will be 58%.

* Completing а repair sets **is\_functional** to **True**.

### Class Smartphone

In the **smartphone.py** file, the class **Smartphone** should be implemented. Asmartphone is a **type** of **base device**, and its **device\_type** property is **set** to **"Smartphone"**.

#### Methods

##### \_\_init\_\_(serial\_number: str, durability: int, is\_functional: bool)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

##### repair()

* If the device **needs repair** (is\_functional=False), the method **increases** its **durability** by **tripling** its **current value**. However, it should **not exceed** the maximum level of **100%**.

Example: If the current durability is 8%, the new value after repair will be 24%.

* Completing а repair sets **is\_functional** to **True**.

### Class Smartwatch

In the **smartwatch.py** file, the class **Smartwatch** should be implemented. A smartwatch is a **type** of **base device**, and its **device\_type** property is **set** to **"Smartwatch"**.

#### Methods

##### \_\_init\_\_(serial\_number: str, durability: int, is\_functional: bool)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

##### repair()

* If the device **needs repair** (is\_functional=False), the method **increases** its **durability** by **doubling** its **current value**. However, it should **not exceed** the maximum level of **100%**.

Example: If the current durability is 8%, the new value after repair will be 16%.

* Completing а repair sets **is\_functional** to **True**.

### Class RepairShop

In the **repair\_shop.py** file, the class **RepairShop** should be implemented. It is a **class** for **all** **types** of **repair shops**, and it **can be instantiated**.

#### Structure

The class should have the following attributes:

* **name:** **str**
  + The value represents the **name** of the **repair shop**.
  + The name must **contain at least two characters** (leading or trailing white spaces do not count). If it doesn't meet the criteria, **raise** a **ValueError** with the following message:  
    **"Invalid repair shop name!"**
* **device\_types:** **tuple**
  + Contains the **device types** that the **shop can accept** for repair.
  + The tuple **must** have **at least one element**, otherwise **raise** a **ValueError** with the following message: **"No device types provided!"**
* **pending\_devices: list**
  + **Initially** set to an **empty list**.
  + It will store a **collection** of **devices** (objects) **accepted** for **repair** in the shop.
* **repaired\_devices: list**
  + **Initially** set to an **empty list**.
  + It will store a **collection** of **devices** (objects) **repaired** in the shop.

#### Methods

##### **\_\_init\_\_(name: str, device\_types: tuple**)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

##### repair()

* **Repairs** all **pending devices** (if any) and **moves** them to the list with **repaired devices**.
  + Note: Use the method[**repair()**](#_repair())for each **device** to guarantee the proper repair processing.
  + **Remove** repaired devicesfromthe **pending** list and **add** them to the **repaired** list.
* **Returns** a string in the following format: **"Repaired {count} device/s."**

##### status()

* **Returns** a string in the following format: **"{repair\_shop\_name} has {pending\_count} devices pending for repair and {repaired\_count} devices repaired."**

### Class TechServiceManager

In the **tech\_service\_manager.py** file, the class **TechServiceManager** should be implemented. It will contain the project's functionality.

#### Structure

The class should have the following attributes:

* **devices: list**
  + Initially, an empty list to store **all device objects** received in the tech service.
* **repair\_shops: list**
  + Initially, an empty list to store **all repair shop objects** working with the tech service.

#### Methods

##### \_\_init\_\_()

* In the **\_\_init\_\_** method, all the needed attributes must be set.

##### add\_device(device\_type: str, serial\_number: str, durability: int, is\_functional: bool)

The method **adds** (upon certain conditions) a **device** of the given type and **adds** it to the manager's **devices** collection:

* **First**, check **if the device type is valid**, and **if not**, **raise** a **ValueError** with the following message: **"Invalid device type!"**
* Then, **create and** **add** the **device** to the manager's **devices** collection:
  + Aftercreating the object, **check** if the device **needs repair** (use the[**check\_functionality()**](#_check_functionality())method)andthen **add it** to themanager's collection of **devices**.
  + **Return** the following message: "**{device\_type} is successfully added."**
* **Valid types** of devices are: **"Laptop"**, **"Smartphone"** and **"Smartwatch"**.
* **Serial numbers** will be **unique** **valid** strings. You are not supposed to check that.

##### add\_repair\_shop(**name: str, device\_types: tuple**)

The method **creates** (upon certain conditions) a **repair** **shop** and **adds** it to the manager's **repair\_shops** collection:

* **First**, check if the **device\_types** tuplecontains **at least one** (or more)of the desired device types **"Laptop"**, **"Smartphone"** or **"Smartwatch"**. If **not**, **raise** a **ValueError** with the following message: "**No valid device type!"**
* Then **create and add** the **shop** to the manager's **repair\_shops** collection. **Return** the following message:

"**{name} is successfully added as a repair shop."**

* **Names** will be **unique** **valid** strings. You are not supposed to check that.

##### send\_for\_repair(repair\_shop\_name: str, device\_type: str)

The method sends a device for repair upon certain conditions. The **repair shop name** and **device type** will be **valid existing values**.

The manager **sends** aspecific **device** of the given type to a **repair** **shop** with theprovided name **if all** pre-defined **conditions** are **met**:

* **First**,find the repair **shop** by its **name** andcheck **if** itcan **accept** this **type** of **device**:
  + If the repair shop **cannot accept** thisdevice,the **repair** is **impossible**.
  + **Return** the message: **"The shop cannot repair this device type."**
* **Then**, Identify the **device** that needs to be sent:
  + Take the **first occurrence** (insertion order) of the given **device type** from the manager's collection of devices **needing repair** (**is\_functional=False**).
  + If there is **no malfunction** in any devices of the given type, **return**:

**"There is no {device\_type} that needs repair."**

* **Finally**, the manager **sends** the **device** tothe **repair shop**.A series of **actions** are **required**:
  + The **device** is **removed** from themanager's **devices** collection.
  + The **device** is **added** to the **repair shop's** collectionof **pending** **devices**.
  + **Return** the following message:

**"{device\_serial\_number} was sent for repair to {repair\_shop\_name}."**

##### process\_repairs(repair\_shop: RepairShop)

The method **repairs** **all devices** **pending** in the given **repair** **shop** object (always valid).

* Each **pending** devicegets **repaired** and **moved** to **repaired devices**.
  + Remember that repairs should **not exceed** the **durability** **maximum** (100%).
  + **Hint**: Use the shop's [**repair()**](#_repair()_1) method.
* **Return** the following message:

**"Repaired {count} device/s."**

##### receive\_repaired\_devices(repair\_shop: RepairShop)

The method **removes** **all repaired devices** (if any) from thegiven **repair shop** (always a valid object) and **adds** them back to the manager's **devices** collection.

* **Return** the following message: **"Received {count} repaired devices."**

##### tech\_service\_status()

The method **displays** detailed **information** about the **tech service** and the **repair** **shops**.

* First, **include** informationabout **devices** from the **manager's collection** andthe **repair shops count**.
* Then, **include** information about repair **shops** (if any)and **their devices**:
  + **Sort** them by the **repair shop name** in **ascending order**. See the [**Examples**](#_Examples).
  + **Hint**: Use the repair shop's [**status()**](#_status()) method.
* The returned information should follow the format of the example below (each repair **shop's info** is on a **new line**):

**"\*\*\*Tech Service\*\*\***

**Total number of functional devices: {managers\_functional\_devices\_count}**

**Total number of malfunctioning devices: {managers\_malfunctioning\_devices\_count}**

**Repair shops count: {repair\_shops\_count}**

**@{repair\_shop\_name1} has {pending\_count} devices pending for repair and {repaired\_count} devices repaired.**

**...**

**@{repair\_shop\_namen} has {pending\_count} devices pending for repair and {repaired\_count} devices repaired."**

### Examples

|  |
| --- |
| **Test Code** |
| ***# Create the service manager***  manager = TechServiceManager()  ***# Add devices***  print(manager.add\_device("Laptop", "LPT1234A", 40, True))  print(manager.add\_device("Laptop", "LPT5678B", 1, True))  print(manager.add\_device("Laptop", "LZT5678B", 2, True))  print(manager.add\_device("Smartphone", "SPH0001X", 10, False))  print(manager.add\_device("Smartphone", "SPH0002Y", 80, True))  print(manager.add\_device("Smartphone", "SYH0002Y", 1, True))  print(manager.add\_device("Smartwatch", "SZT3009Z", 5, False))  print(manager.add\_device("Smartwatch", "SWT3009Z", 5, False))  print()  ***# Add repair shops***  print(manager.add\_repair\_shop("BFixIt Center", ("Desktop", "Laptop", "Smartphone", "Smartwatch")))  print(manager.add\_repair\_shop("AQuickFix", ("HomeAI", "Smartwatch")))  print()  ***# Send devices for repair***  print(manager.send\_for\_repair("BFixIt Center", "Laptop"))  print(manager.send\_for\_repair("BFixIt Center", "Smartphone"))  print(manager.send\_for\_repair("BFixIt Center", "Smartwatch"))  print(manager.send\_for\_repair("BFixIt Center", "Laptop"))  print(manager.send\_for\_repair("AQuickFix", "Smartphone"))  print(manager.send\_for\_repair("AQuickFix", "Smartwatch"))  print()  ***# Process repairs at first shop***  print(manager.process\_repairs(manager.repair\_shops[0]))  print(manager.process\_repairs(manager.repair\_shops[0]))  print()  ***# Display current service status***  print(manager.tech\_service\_status())  print()  ***# Receive repaired devices***  print(manager.receive\_repaired\_devices(manager.repair\_shops[0]))  print(manager.receive\_repaired\_devices(manager.repair\_shops[1]))  print()  ***# Display final service status***  print(manager.tech\_service\_status()) |
| **Output** |
| Laptop is successfully added.  Laptop is successfully added.  Laptop is successfully added.  Smartphone is successfully added.  Smartphone is successfully added.  Smartphone is successfully added.  Smartwatch is successfully added.  Smartwatch is successfully added.  BFixIt Center is successfully added as a repair shop.  AQuickFix is successfully added as a repair shop.  LPT5678B was sent for repair to BFixIt Center.  SPH0001X was sent for repair to BFixIt Center.  SZT3009Z was sent for repair to BFixIt Center.  There is no Laptop that needs repair.  The shop cannot repair this device type.  SWT3009Z was sent for repair to AQuickFix.  Repaired 3 device/s.  Repaired 0 device/s.  \*\*\*Tech Service\*\*\*  Total number of functional devices: 3  Total number of malfunctioning devices: 1  Repair shops count: 2  @AQuickFix has 1 devices pending for repair and 0 devices repaired.  @BFixIt Center has 0 devices pending for repair and 3 devices repaired.  Received 3 repaired devices.  Received 0 repaired devices.  \*\*\*Tech Service\*\*\*  Total number of functional devices: 6  Total number of malfunctioning devices: 1  Repair shops count: 2  @AQuickFix has 1 devices pending for repair and 0 devices repaired.  @BFixIt Center has 0 devices pending for repair and 0 devices repaired. |

## Problem 3: Unit Testing (100 points)

You will **be provided with another skeleton** for this problem. **Open** the **new skeleton** as a **new project** and write tests for the **Volcano** class. The class will have some methods, fields, and one constructor, all of them working properly. You are **NOT ALLOWED** to change anything in the class code. Cover the whole class with unit tests to make sure that the class is working as intended. Submit **only** the **test** folder.