Assignment 4

```
'''1
• Create classes Employee, Manager, and Developer.
• Classes Manager and Developer inherits Employee.
• Create appropriate methods to get details and display det
ails for all classes.
• Implement constructors and destructors for same.
• Modify the Employee class to include a method that calcul
ates a bonus based on the salary.
• Modify the Manager and Developer classes to utilize this
method.
• Add a new class called Intern that also inherits from Emp
loyee.
• The Intern class should have an additional attribute dura
tion (in months).
• Implement methods to display intern details and calculate
a stipend based on the duration.
'''

class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        print(f"Employee {self.name} created.")

    def __del__(self):
        print(f"Employee {self.name} deleted.")

    def get_details(self):
        return f"Name: {self.name}, Salary: {self.salary}"

    def display_details(self):
        print(self.get_details())

    def calculate_bonus(self):
```

```
        return self.salary * 0.10  # 10% bonus


class Manager(Employee):
    def __init__(self, name, salary, team_size):
        super().__init__(name, salary)
        self.team_size = team_size
        print(f"Manager {self.name} created with team size
{self.team_size}.")

    def __del__(self):
        print(f"Manager {self.name} deleted.")

    def get_details(self):
        return f"{super().get_details()}, Team Size: {self.
team_size}"

    def display_details(self):
        print(self.get_details())
        bonus = self.calculate_bonus()
        print(f"Bonus: {bonus}")


class Developer(Employee):
    def __init__(self, name, salary, programming_language):
        super().__init__(name, salary)
        self.programming_language = programming_language
        print(f"Developer {self.name} created, programming
in {self.programming_language}.")

    def __del__(self):
        print(f"Developer {self.name} deleted.")

    def get_details(self):
        return f"{super().get_details()}, Programming Langu
age: {self.programming_language}"

    def display_details(self):
```

```
        print(self.get_details())
        bonus = self.calculate_bonus()
        print(f"Bonus: {bonus}")


class Intern(Employee):
    def __init__(self, name, salary, duration):
        super().__init__(name, salary)
        self.duration = duration
        print(f"Intern {self.name} created for {self.durati
on} months.")

    def __del__(self):
        print(f"Intern {self.name} deleted.")

    def get_details(self):
        return f"{super().get_details()}, Duration: {self.d
uration} months"

    def display_details(self):
        print(self.get_details())
        stipend = self.calculate_stipend()
        print(f"Stipend: {stipend}")

    def calculate_stipend(self):
        return self.salary * 0.05 * self.duration  # 5% of
salary per month


emp = Employee("Alice", 50000)
emp.display_details()

mgr = Manager("Bob", 80000, 5)
mgr.display_details()

dev = Developer("Charlie", 60000, "Python")
dev.display_details()
```

```
intern = Intern("Dave", 20000, 6)
intern.display_details()
```

```
'''2
• Create classes Person, Student, and Result.
• Classes: Person (base class), Student (inherits from Pers
on), and Result (inherits from
Student).
• Create appropriate methods to get and display details for
all classes.
• Implement constructors and destructors for each class.
• Modify the Result class to include a method that calculat
es and displays the highest and
lowest marks obtained by the student.
• Add a new class called HonorsResult that also inherits fr
om Result.
• The HonorsResult class should have an additional attribut
e for honors classification (e.g.,
"First Class", "Second Class", etc.).
• Implement methods to display the honors classification an
d determine if the
• student qualifies for honors based on their average mark
s.
'''


class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print(f"Person {self.name} created.")

    def display_details(self):
        print(f"Name: {self.name}, Age: {self.age}")

    def __del__(self):
        print(f"Person {self.name} destroyed.")
```

```
class Student(Person):
    def __init__(self, name, age, student_id):
        super().__init__(name, age)
        self.student_id = student_id
        print(f"Student {self.name} with ID {self.student_i
d} created.")

    def display_details(self):
        super().display_details()
        print(f"Student ID: {self.student_id}")

    def __del__(self):
        print(f"Student {self.name} destroyed.")


class Result(Student):
    def __init__(self, name, age, student_id, marks):
        super().__init__(name, age, student_id)
        self.marks = marks
        print(f"Result for {self.name} created.")

    def calculate_highest_lowest(self):
        highest = max(self.marks)
        lowest = min(self.marks)
        print(f"Highest Marks: {highest}, Lowest Marks: {lo
west}")

    def display_details(self):
        super().display_details()
        self.calculate_highest_lowest()

    def __del__(self):
        print(f"Result for {self.name} destroyed.")


class HonorsResult(Result):
    def __init__(self, name, age, student_id, marks, classi
```

```
fication):
        super().__init__(name, age, student_id, marks)
        self.classification = classification
        print(f"Honors Result for {self.name} created.")

    def display_honors_classification(self):
        average_marks = sum(self.marks) / len(self.marks)
        print(f"Honors Classification: {self.classificatio
n}")
        print(f"Average Marks: {average_marks}")
        if average_marks >= 70:
            print(f"{self.name} qualifies for Honors.")
        else:
            print(f"{self.name} does not qualify for Honor
s.")

    def display_details(self):
        super().display_details()
        self.display_honors_classification()

    def __del__(self):
        print(f"Honors Result for {self.name} destroyed.")



if __name__ == "__main__":
    student1 = HonorsResult("Alice", 20, "S123", [85, 78, 9
2, 88], "First Class")

    student1.display_details()

    del student1
```

```
'''3
Create classes Device, Smartphone, and FeaturePhone.
• Classes:
• Device (base class)
• Smartphone (inherits from Device)
```

```
• FeaturePhone (inherits from Device)
• Create appropriate methods to get and display details for
all classes.
• Implement constructors and destructors for each class.
• Modify the Smartphone class to include a method that calc
ulates and displays the available
storage space based on the total storage and used storage.
• Modify the FeaturePhone class to include a method that ca
lculates and displays the battery
life based on the battery capacity and usage rate.
• Add a new class called GamingSmartphone that also inherit
s from Smartphone.
• The GamingSmartphone class should have an additional attr
ibute for gaming-specific
features (e.g., "High Refresh Rate", "Enhanced Cooling").
• Implement methods to display gaming features and determin
e if the smartphone is suitable
for gaming based on these features.
'''


class Device:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model
        print(f"Device {self.brand} {self.model} created.")

    def display_details(self):
        print(f"Brand: {self.brand}, Model: {self.model}")

    def __del__(self):
        print(f"Device {self.brand} {self.model} destroye
d.")


class Smartphone(Device):
    def __init__(self, brand, model, total_storage, used_st
orage):
```

```
        super().__init__(brand, model)
        self.total_storage = total_storage
        self.used_storage = used_storage
        print(f"Smartphone {self.brand} {self.model} create
d.")

    def display_details(self):
        super().display_details()
        self.calculate_available_storage()

    def calculate_available_storage(self):
        available_storage = self.total_storage - self.used_
storage
        print(f"Total Storage: {self.total_storage}GB, Used
Storage: {self.used_storage}GB, Available Storage: {availab
le_storage}GB")

    def __del__(self):
        print(f"Smartphone {self.brand} {self.model} destro
yed.")


class FeaturePhone(Device):
    def __init__(self, brand, model, battery_capacity, usag
e_rate):
        super().__init__(brand, model)
        self.battery_capacity = battery_capacity
        self.usage_rate = usage_rate
        print(f"FeaturePhone {self.brand} {self.model} crea
ted.")

    def display_details(self):
        super().display_details()
        self.calculate_battery_life()

    def calculate_battery_life(self):
        battery_life = self.battery_capacity / self.usage_r
ate
```

```
        print(f"Battery Capacity: {self.battery_capacity}mA
h, Usage Rate: {self.usage_rate}mAh/hour, Estimated Battery
Life: {battery_life} hours")

    def __del__(self):
        print(f"FeaturePhone {self.brand} {self.model} dest
royed.")


class GamingSmartphone(Smartphone):
    def __init__(self, brand, model, total_storage, used_st
orage, gaming_features):
        super().__init__(brand, model, total_storage, used_
storage)
        self.gaming_features = gaming_features
        print(f"GamingSmartphone {self.brand} {self.model}
created.")

    def display_details(self):
        super().display_details()
        self.display_gaming_features()
        self.is_suitable_for_gaming()

    def display_gaming_features(self):
        print(f"Gaming Features: {', '.join(self.gaming_fea
tures)}")

    def is_suitable_for_gaming(self):
        if "High Refresh Rate" in self.gaming_features and
"Enhanced Cooling" in self.gaming_features:
            print(f"{self.brand} {self.model} is suitable f
or gaming.")
        else:
            print(f"{self.brand} {self.model} is not ideal
for gaming.")

    def __del__(self):
        print(f"GamingSmartphone {self.brand} {self.model}
```

```
destroyed.")


# Example usage
if __name__ == "__main__":

    gaming_phone = GamingSmartphone("SuperTech", "GamerX",
256, 120, ["High Refresh Rate", "Enhanced Cooling", "RGB Li
ghting"])


    gaming_phone.display_details()

    print("\n")


    feature_phone = FeaturePhone("OldSchool", "Classic200",
1500, 50)


    feature_phone.display_details()

    del gaming_phone
    del feature_phone
```

```
'''4
Write a function that calculates the interest rate for a sa
vings account. The interest is calculated as
interest = (balance * rate) / months. Use try-except to han
dle the case where the number of months
is zero, which would raise a ZeroDivisionError.
'''

def calculate_interest(balance, rate, months):
    try:
        interest = (balance * rate) / months
        return interest
    except ZeroDivisionError:
```

```
        return "Error: Number of months cannot be zero."


balance = 1000
rate = 0.05
months = 0

interest = calculate_interest(balance, rate, months)
print(interest)
```

```
'''5
Write a temperature conversion function that converts Celsi
us to Fahrenheit. If the user inputs a
temperature below absolute zero (−273.15°C), raise a custom
BelowAbsoluteZeroError. Catch and
handle this exception with a message indicating that the te
mperature is invalid.
'''

class BelowAbsoluteZeroError(Exception):
    pass

def celsius_to_fahrenheit(celsius):
    try:
        if celsius < -273.15:
            raise BelowAbsoluteZeroError("Temperature is be
low absolute zero (-273.15°C).")
        fahrenheit = (celsius * 9/5) + 32
        return fahrenheit
    except BelowAbsoluteZeroError as e:
        return f"Error: {e}"


celsius = -300
result = celsius_to_fahrenheit(celsius)
print(result)
```

```
'''6
In a flight booking system, write a function that accepts u
ser input for seat selection (like "A5",
"B7", etc.). Raise a ValueError if the user enters an inval
id seat format (e.g., "5A" or "Z99").
Handle the exception and prompt the user to enter a valid s
eat number.
'''

import re

def seat_selection():
    while True:
        try:
            seat = input("Please enter a seat (e.g., A5, B
7): ").upper()
            if not re.match(r'^[A-H][1-9]\d?$', seat):
                raise ValueError("Invalid seat format. Seat
must be a letter followed by a number.")

            print(f"Seat {seat} selected successfully.")
            break

        except ValueError as e:
            print(e)


seat_selection()
```

```
'''7
In an e-commerce application, create a function that accept
s the number of items a customer wants
to purchase and checks if the stock is sufficient. Raise a
StockError (custom exception) if the
requested quantity exceeds the available stock. Catch this
exception and notify the user to adjust
their order.
```

```
'''



class StockError(Exception):
    pass


def purchase_items(available_stock):
    while True:
        try:

            quantity = int(input("Enter the number of items
you want to purchase: "))


            if quantity > available_stock:
                raise StockError(f"Only {available_stock} i
tems available. Please adjust your order.")

            print(f"Purchase successful! You ordered {quant
ity} item(s).")
            break

        except StockError as e:
            print(e)

        except ValueError:
            print("Invalid input. Please enter a valid numb
er.")


available_stock = 10
purchase_items(available_stock)
```