**Python OOPs Task Sheet - Single Inheritance Problems**

**Problem 1: Smart Savings Account**

**Problem Statement:**  
Create a class Account that has a constructor to initialize balance and a method deposit(amount) to add money.  
Create a subclass SavingsAccount that overrides the deposit method to add a 2% bonus on every deposit.  
The bonus should be applied only if the deposit amount is greater than 1000.

**Requirements:**

* Use super() to reuse the parent deposit logic.
* Implement a method show\_balance() to return the current balance.

**Sample Input:**

acc = SavingsAccount(5000)

acc.deposit(1500)

print(acc.show\_balance())

**Expected Output:**

Balance after bonus deposit: 6530.0

**Problem 2: Celsius to Weather Reporter**

**Problem Statement:**  
Create a class Celsius that stores a temperature and has a method to\_fahrenheit() for conversion.  
Create a subclass WeatherReport that:

* Overrides the constructor to add a location attribute.
* Adds a report() method to print temperature in both Celsius and Fahrenheit.

**Challenge:**

* Validate that temperature is not below -273.15.
* Ensure the child constructor calls the parent constructor.

**Sample Input:**

w = WeatherReport("Dubai", 45)

w.report()

**Expected Output:**

Location: Dubai

Temp in Celsius: 45

Temp in Fahrenheit: 113.0

**Problem 3: Department Library Book Tracker**

**Problem Statement:**  
Create a base class Library that holds a total number of books (say 100).  
Create a subclass DepartmentLibrary that holds its own book count (say 30) and shadows the books variable.

Create a method display() that:

* Shows total books from the base class.
* Shows department books from the subclass.

**Bonus Challenge:**

* Access and print the base class books even after it's shadowed.

**Problem 4: Special Number Checker**

**Problem Statement:**  
Create a class Number with a method is\_even() to check if a number is even.  
Create a subclass SpecialNumber that:

* Inherits from Number
* Implements double\_if\_even() which doubles the number if it is even.

**Requirements:**

* Add a reset(n) method to set a new number.
* Ensure zero is treated as even.

**Sample Test:**

n = SpecialNumber(4)

print(n.double\_if\_even()) # 8

n.reset(5)

print(n.double\_if\_even()) # 5

**Problem 5: Employee Role Chain**

**Problem Statement:**  
Create a class Employee with an attribute name and method get\_role() returning "Employee".  
Create a subclass Developer that overrides get\_role() to return "Name is a Developer".  
Add a method describe() that prints both the base and derived roles.

**Sample Input:**

d = Developer("Alice")

d.describe()

**Expected Output:**

Base Role: Employee

Current Role: Alice is a Developer