Java OOPs Practice Sheet with Solutions

Q1: Create a Bank Account Class

Create a class BankAccount with fields: accountNumber, accountHolderName, and balance.
Methods:
- deposit(double amount)
- withdraw(double amount)
- displayDetails()

Solution:

```
class BankAccount {
    String accountHolderName;
    String accountNumber;
    double balance;

    BankAccount(String name, String number, double initialBalance) {
        accountHolderName = name;
        accountNumber = number;
        balance = initialBalance;
    }

    void deposit(double amount) {
        balance += amount;
    }

    void withdraw(double amount) {
        if (amount <= balance) balance -= amount;
    }

    void displayDetails() {
        System.out.println("Account Holder: " + accountHolderName);
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Balance: " + balance);
    }
}
```

Q2: Inheritance Example with Person and Employee

Create a base class Person and subclass Employee which adds company name.

Solution:

```
class Person {
    String name;
    int age;
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

Java OOPs Practice Sheet with Solutions

```
class Employee extends Person {
    String company;
    Employee(String name, int age, String company) {
        super(name, age);
        this.company = company;
    }
}
```

Q3: Method Overriding (Polymorphism) Example

Demonstrate runtime polymorphism using Animal and Dog class.

```
class Animal {
    void sound() {
        System.out.println("Animal makes sound");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Dog barks");
    }
}

Main:
Animal a = new Dog();
a.sound();  // Output: Dog barks
```

Q4: Interface and Abstraction

Create interface Vehicle with move(). Implement it in Car and Bike.

```
interface Vehicle {
    void move();
}

class Car implements Vehicle {
    public void move() {
        System.out.println("Car moves");
    }
}
```

Q5: Encapsulation using Private Fields

Create class Student with private fields and getter/setter methods.

```
class Student {
    private String name;
    private int age;
    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        if (age > 0) this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```