Write a Java program that creates a class hierarchy for employees of a company. The base class should be Employee, with subclasses Manager, Developer, and Programmer. Each subclass should have properties such as name, address, salary, and job title. Implement methods for calculating bonuses, generating performance reports, and managing projects.

Sample Solution:

Java Code:

```
// Class declaration for Employee
class Employee {
    // Private fields for the Employee class
    private String name;
    private String address;
    private double salary;
    private String jobTitle;

    // Constructor for the Employee class
    public Employee(String name, String address, double salary, String jobT
        // Initializing the name field
        this.name = name;
        // Initializing the address field
        this.address = address;
        // Initializing the salary field
        this.salary = salary;
        // Initializing the jobTitle field
        this.jobTitle = jobTitle;
    }

    // Getter method for the name field
    public String getName() {
        return name;
    }

    // Getter method for the address field
    public String getAddress() {
        return address;
    }

    // Getter method for the salary field
    public double getSalary() {
```

```
        return salary;
    }

    // Getter method for the jobTitle field
    public String getJobTitle() {
        return jobTitle;
    }

    // Method to calculate the bonus for the employee
    public double calculateBonus() {
        // Default implementation for bonus calculation
        return 0.0;
    }

    // Method to generate the performance report for the employee
    public String generatePerformanceReport() {
        // Default implementation for performance report
        return "No performance report available.";
    }
}
```

Explanation:

- Employee class: This class represents a generic employee with private instance variables 'name', 'address', 'salary', and 'jobTitle'. It also provides getter methods to access these private variables.
    - getName(): Returns the employee's name.
    - getAddress(): Returns the employee's address.
    - getSalary(): Returns the employee's salary.
    - getJobTitle(): Returns the employee's job title.
- calculateBonus(): This method is used to calculate the bonus for an employee. In the base class, it provides a default implementation that returns 0.0. Subclasses can override this method to provide custom bonus calculation logic.
- generatePerformanceReport(): This method generates a performance report for an employee. Similar to the bonus calculation, it provides a default implementation that returns "No performance report available." Subclasses can override this method to provide custom performance report generation logic.

This class is designed to be extended by subclasses like "Manager", "Developer", and "Programmer", which can provide their own implementations of bonus calculation and performance report generation as per their specific roles and responsibilities.

```
// Class declaration for Manager which extends Employee
class Manager extends Employee {
    // Private field for the number of subordinates
    private int numberOfSubordinates;

    // Constructor for the Manager class
    public Manager(String name, String address, double salary, String jobT
        // Calling the constructor of the superclass Employee
        super(name, address, salary, jobTitle);
        // Initializing the numberOfSubordinates field
        this.numberOfSubordinates = numberOfSubordinates;
    }

    // Getter method for the numberOfSubordinates field
    public int getNumberOfSubordinates() {
        return numberOfSubordinates;
    }

    // Overridden method to calculate the bonus for the manager
    @Override
    public double calculateBonus() {
        // Custom implementation for bonus calculation for managers
        return getSalary() * 0.15;
    }

    // Overridden method to generate the performance report for the manager
    @Override
    public String generatePerformanceReport() {
        // Custom implementation for performance report for managers
        return "Performance report for Manager " + getName() + ": Excellent
    }

    // Custom method for managing projects
    public void manageProject() {
        // Printing a message indicating the manager is managing a project
        System.out.println("Manager " + getName() + " is managing a project
    }
```

```
    }
```

Explanation:

- extends Employee: This line indicates that the "Manager" class inherits from the "Employee" class. It means that a Manager is a specialized type of Employee and inherits all the attributes and methods of the Employee class.
- private int numberOfSubordinates: This instance variable represents the number of subordinates managed by the manager. It is specific to the "Manager" class and not present in the base "Employee" class.
- public Manager(String name, String address, double salary, String jobTitle, int numberOfSubordinates): This is the constructor for the "Manager" class. It takes parameters for 'name', 'address', 'salary', 'jobTitle', and numberOfSubordinates, which are used to initialize the attributes inherited from the "Employee" class as well as the numberOfSubordinates specific to managers. The super(...) keyword is used to call the constructor of the superclass (Employee) to initialize its attributes.
- public int getNumberOfSubordinates(): This method allows you to retrieve the number of subordinates managed by the manager.
- @Override public double calculateBonus(): This method is marked with the @Override annotation, indicating that it is an overridden method from the superclass (Employee). The "calculateBonus()" method provides a custom implementation for bonus calculation for managers. In this case, it calculates the bonus as 15% of the manager's salary.
- @Override public String generatePerformanceReport(): Similar to the "calculateBonus()" method, this method is also marked as an override and provides a custom implementation for generating a performance report for managers. It returns a specific performance report message for managers, including the manager's name and an "Excellent" rating.
- public void manageProject(): This is a custom method specific to the "Manager" class. It simulates the action of a manager managing a project by printing a message to the console.

```
    // Class declaration for Developer which extends Employee
    class Developer extends Employee {
        // Private field for the programming language
        private String programmingLanguage;

        // Constructor for the Developer class
        public Developer(String name, String address, double salary, String job
            // Calling the constructor of the superclass Employee
            super(name, address, salary, jobTitle);
            // Initializing the programmingLanguage field
```

```
        this.programmingLanguage = programmingLanguage;
    }

    // Getter method for the programmingLanguage field
    public String getProgrammingLanguage() {
        return programmingLanguage;
    }

    // Overridden method to calculate the bonus for the developer
    @Override
    public double calculateBonus() {
        // Custom implementation for bonus calculation for developers
        return getSalary() * 0.10;
    }

    // Overridden method to generate the performance report for the develop
    @Override
    public String generatePerformanceReport() {
        // Custom implementation for performance report for developers
        return "Performance report for Developer " + getName() + ": Good";
    }

    // Custom method for writing code
    public void writeCode() {
        // Printing a message indicating the developer is writing code
        System.out.println("Developer " + getName() + " is writing code in
    }
}
```

Explanation:

- extends Employee: Similar to the "Manager" class, this line indicates that the Developer class inherits from the "Employee" class. It means that a 'Developer' is a specialized type of 'Employee' and inherits all the attributes and methods of the Employee class.
- private String programmingLanguage: This instance variable represents the programming language that the developer specializes in. It is specific to the "Developer" class and not present in the base "Employee" class.
- public Developer(String name, String address, double salary, String jobTitle, String programmingLanguage): This is the constructor for the "Developer" class. It takes parameters for 'name', 'address', 'salary', 'jobTitle', and programmingLanguage, which are used to initialize the

attributes inherited from the "Employee" class as well as the programmingLanguage specific to developers. The super(...) keyword is used to call the constructor of the superclass (Employee) to initialize its attributes.

- public String getProgrammingLanguage(): This method allows you to retrieve the programming language specialization of the developer.
- @Override public double calculateBonus(): This method is marked with the @Override annotation, indicating that it is an overridden method from the superclass (Employee). The "calculateBonus()" method provides a custom implementation for bonus calculation for developers. In this case, it calculates the bonus as 10% of the developer's salary.
- @Override public String generatePerformanceReport(): Similar to the "calculateBonus()" method, this method is also marked as an override and provides a custom implementation for generating a performance report for developers. It returns a specific performance report message for developers, including the developer's name and a "Good" rating.
- public void writeCode(): This is a custom method specific to the "Developer" class. It simulates the action of a developer writing code in their specialized programming language by printing a message to the console.

```
    // Class declaration for Programmer which extends Developer
    class Programmer extends Developer {
        // Constructor for the Programmer class
        public Programmer(String name, String address, double salary, String pr
            // Calling the constructor of the superclass Developer
            super(name, address, salary, "Programmer", programmingLanguage);
        }

        // Overridden method to calculate the bonus for the programmer
        @Override
        public double calculateBonus() {
            // Custom implementation for bonus calculation for programmers
            return getSalary() * 0.12;
        }

        // Overridden method to generate the performance report for the progra
        @Override
        public String generatePerformanceReport() {
            // Custom implementation for performance report for programmers
            return "Performance report for Programmer " + getName() + ": Excel
        }
```

```
        // Custom method for debugging code
        public void debugCode() {
            // Printing a message indicating the programmer is debugging code
            System.out.println("Programmer " + getName() + " is debugging code
        }
    }
```

Explanation:

- extends Developer: This line indicates that the "Programmer" class inherits from the "Developer" class. It means that a 'Programmer' is a specialized type of 'Developer' and inherits all the attributes and methods of the Developer class.
- public Programmer(String name, String address, double salary, String programmingLanguage): This is the constructor for the "Programmer" class. It takes parameters for 'name', 'address', 'salary', and 'programmingLanguage'. It passes these parameters to the constructor of the superclass (Developer) using the super(...) keyword to initialize the attributes inherited from the "Developer" class. The 'jobTitle' parameter is set to "Programmer" to indicate the specific job title for programmers.
- @Override public double calculateBonus(): This method is marked with the @Override annotation, indicating that it is an overridden method from the superclass (Developer). The "calculateBonus()" method provides a custom implementation for bonus calculation for programmers. In this case, it calculates the bonus as 12% of the programmer's salary.
- @Override public String generatePerformanceReport(): Similar to the "calculateBonus()" method, this method is also marked as an override and provides a custom implementation for generating a performance report for programmers. It returns a specific performance report message for programmers, including the programmer's name and an "Excellent" rating.
- public void debugCode(): This is a custom method specific to the "Programmer" class. It simulates the action of a programmer debugging code in their specialized programming language by printing a message to the console.

```
    // Public class declaration for Main
    public class Main {
        // Main method
        public static void main(String[] args) {
            // Creating an instance of Manager
            Manager manager = new Manager("Avril Aroldo", "1 ABC St", 80000.0,
            // Creating an instance of Developer
            Developer developer = new Developer("Iver Dipali", "2 PQR St", 720
            // Creating an instance of Programmer
            Programmer programmer = new Programmer("Yaron Gabriel", "3 ABC St",
```

```
            // Printing the manager's bonus
            System.out.println("Manager's Bonus: $" + manager.calculateBonus()
            // Printing the developer's bonus
            System.out.println("Developer's Bonus: $" + developer.calculateBonu
            // Printing the programmer's bonus
            System.out.println("Programmer's Bonus: $" + programmer.calculateBo

            // Printing the manager's performance report
            System.out.println(manager.generatePerformanceReport());
            // Printing the developer's performance report
            System.out.println(developer.generatePerformanceReport());
            // Printing the programmer's performance report
            System.out.println(programmer.generatePerformanceReport());

            // Manager managing a project
            manager.manageProject();
            // Developer writing code
            developer.writeCode();
            // Programmer debugging code
            programmer.debugCode();
        }
    }
```

Explanation:

- Creating Employee Objects:
    - Three employee objects are created: 'manager', 'developer', and 'programmer', each with their specific attributes such as name, address, salary, and job title.
    - manager is an instance of the "Manager" class.
    - developer is an instance of the "Developer" class.
    - programmer is an instance of the "Programmer" class.
- Calculating Bonuses:
    - The program calls the "calculateBonus()" method for each employee type (manager, developer, and programmer) to calculate their respective bonuses.
    - The bonuses are displayed on the console.
- Generating Performance Reports:

    - The program calls the "generatePerformanceReport()" method for each employee type (manager, developer, and programmer) to generate performance reports.
    - The performance reports are displayed on the console.
- Specific Actions:
    - For each employee type, specific actions are performed:
        - manager uses the "manageProject()" method to simulate managing a project.
        - developer uses the "writeCode()" method to simulate writing code in a specific programming language.
        - programmer uses the "debugCode()" method to simulate debugging code in a specific programming language.
    - These actions are displayed on the console.

Output:

```
Manager's Bonus: $12000.0
Developer's Bonus: $7200.0
Programmer's Bonus: $9120.0
Performance report for Manager Avril Aroldo: Excellent
Performance report for Developer Iver Dipali: Good
Performance report for Programmer Yaron Gabriel: Excellent
Manager Avril Aroldo is managing a project.
Developer Iver Dipali is writing code in Java
Programmer Yaron Gabriel is debugging code in Python
```

Flowchart of Employee class: