```java
class Volunteer extends Employee {
   public Volunteer(String id) {
      super(id);
   }

   public boolean isPayday(int dayOfMonth) throws IllegalArgumentException {
      if(dayOfMonth < 1 || dayOfMonth > 30) {
         throw new IllegalArgumentException("Day of Month should be a valid integer");
      }
      return false;
   }

   public double calculatePay() throws UnpayableEmployeeException {
      throw new UnpayableEmployeeException("Volunteers don't receive any payment");
   }

   public double calculateDeductions() {
      return 0;
   }
}
```

```java
import java.lang.Exception;
class UnpayableEmployeeException extends Exception {

    public UnpayableEmployeeException(String message) {
        super(message);
    }
}
```

```java
import java.util.*;
class PayrollDispositionImpl implements PayrollDisposition {
    private Map<Employee, Double> payments;

    public PayrollDispositionImpl() {
        payments = new HashMap<>();
    }

    public void sendPayment(Employee empl, double payment) throws NullPointerException, IllegalArgumentException {
        if(empl == null) {
            throw new NullPointerException("Employee can't be null");
        }
        else if(payment <= 0.0) {
            throw new IllegalArgumentException("Payment can't be zero or less");
        }
        payments.put(empl, payment);
    }

    public double getTotal() {
        double t = 0;
        for (double i : payments.values()){
            t += i;
        }
        return t;
    }

    public double getAverage() {
        if(payments.size() > 0) {
            return getTotal() / payments.size();
        }
            return 0;
    }

    public Map<Employee, Double> getPayments() {
        return payments;
    }
}
```

```java
interface PayrollDisposition {
   abstract void sendPayment(Employee empl, double payment) throws IllegalArgumentException;
}
```

```java
import java.util.List;
interface PayrollDB {
   abstract public List<Employee> getEmployeeList();
}
```

```java
import java.util.Map;
class Payroll {
   private int payday;
   private PayrollDisposition disposition;

   public Payroll(PayrollDisposition disposition, int payday) throws NullPointerException, IllegalArgumentException {
      if(disposition == null) {
         throw new NullPointerException("Disposition can't be null");
      }
      if(payday < 1 || payday > 30) {
         throw new IllegalArgumentException("Payday should be valid Integer");
      }
      this.disposition = disposition;
      this.payday = payday;
   }

   public void doPayroll(PayrollDB db) {
      for(Employee i: db.getEmployeeList()) {
         if(i.isPayday(payday)) {
            try {
               disposition.sendPayment(i, i.calculatePay() - i.calculateDeductions());
            }
            catch(Exception e) {}
         }
      }
   }
}
```

```java
abstract class Employee {
    protected String id;

    public Employee(String id) throws NullPointerException, IllegalArgumentException {
        if(id == null) {
            throw new NullPointerException("Employee must have an ID");
        }
        if(id.equals("")) {
            throw new IllegalArgumentException("Employee ID can't be empty");
        }
        this.id = id;
    }

    public String getId() {
        return id;
    }

    abstract boolean isPayday(int dayOfMonth) throws IllegalArgumentException;
    abstract double calculatePay() throws UnpayableEmployeeException;
    abstract double calculateDeductions();
}
```

```java
class Appointee extends Employee {
   private int payday;
   private int hoursPerMonth;
   private double payPerHour;

   public Appointee( String id, int payday, int hoursPerMonth, double payPerHour) throws IllegalArgumentException {
      super(id);
      if(payPerHour <= 0.0 || payday < 1 || payday > 30 || hoursPerMonth <= 0) {
         throw new IllegalArgumentException("faulty parameters were given to the Appointee constructor");
      }
      this.payday = payday;
      this.hoursPerMonth = hoursPerMonth;
      this.payPerHour = payPerHour;
   }

   public boolean isPayday(int dayOfMonth) throws IllegalArgumentException {
      if(dayOfMonth < 1 || dayOfMonth > 30) {
         throw new IllegalArgumentException("Day of Month should be a valid number");
      }
      if(payday == dayOfMonth) {
         return true;
      }
      return false;
   }

   public double calculatePay() {
      return hoursPerMonth * payPerHour;
   }

   public double calculateDeductions() {
      return calculatePay() * 0.4;
   }
}
```