Problems on Exception Handling in Java

---

1. Banking System – Division by Zero (ArithmeticException)

Scenario: A bank wants to calculate the average balance per customer.
Requirements:

- Class `Bank`
  - Fields: `double totalBalance`, `int numberOfCustomers`.
  - Constructor: `(totalBalance, numberOfCustomers)`
  - Method: `double averageBalance()` → returns `totalBalance / numberOfCustomers`.
- Handle `ArithmeticException` when `numberOfCustomers` is 0.
- In `Main`, test with both valid and invalid values.

---

2. Student Marks – ArrayIndexOutOfBoundsException

Scenario: A student marks system stores marks in an array.
Requirements:

- Class `Student`
  - Fields: `String name`, `int[] marks`.
  - Constructor: `(name, marks)`
  - Method: `int getMark(int index)` → returns marks at index.
- Handle `ArrayIndexOutOfBoundsException` if the index is invalid.
- In `Main`, create a student and try to access a valid and invalid index.

---

3. Library System – NullPointerException

Scenario: A library system may sometimes have missing book details.
Requirements:

- Class `Book`
  - Field: `String title`.
  - Constructor: `(title)`
  - Method: `void printTitle()` → prints the title.
- Handle `NullPointerException` if `title` is null.
- In `Main`, create books with and without title, and call `printTitle()`.

---

4. Age Input – InputMismatchException

Scenario: A program asks for user's age using Scanner.
Requirements:

- Class `Person`
  - Field: `int age`.
  - Method: `void readAge()` → use `Scanner` to input age.
  - Handle `InputMismatchException` when the user enters invalid data.
- In `Main`, test with correct and incorrect input.

---

5. Online Order – NumberFormatException

Scenario: An online shop system takes quantity input as String.
Requirements:

- Class `Order`
  - Field: `String quantityInput`.
  - Constructor: `(quantityInput)`
  - Method: `int getQuantity()` → converts `quantityInput` to int.
- Handle `NumberFormatException` if the input is not a number.
- In `Main`, test with both numeric and non-numeric inputs.

---

6. ATM Withdrawal – throw & unchecked exception

Scenario: An ATM allows withdrawals but not more than the balance.
Requirements:

- Class `ATM`
  - Field: `double balance`.
  - Constructor: `(balance)`
  - Method: `void withdraw(double amount)` → if amount > balance, `throw new ArithmeticException("Insufficient funds")`.
- In `Main`, demonstrate valid and invalid withdrawals.

---

7. File Reader – throws & checked exception

Scenario: A file reading system.
Requirements:

- Class `FileReaderDemo`
  - Method: `void readFile(String fileName) throws IOException` → open a file using `FileReader` and read content.
- In `Main`, call `readFile` with a valid and invalid filename.
- Handle exception using `try-catch`.

---

8. Exam Grading – User Defined Exception

Scenario: Grades must be between 0 and 100.
Requirements:

- Class `InvalidGradeException` extends `Exception`.
- Class `Exam`
  - Field: `double grade`.
  - Method: `void setGrade(double g) throws InvalidGradeException` → if g < 0 or g > 100, throw exception.
- In `Main`, test with valid and invalid grades.

---

9. Employee Bonus – Multiple Exception Handling

Scenario: Bonus depends on performance rating.
Requirements:

- Class `Employee`
  - Fields: `String name`, `String rating`.
  - Constructor: `(name, rating)`
  - Method: `int getBonus()` → convert rating to int, then multiply by 1000.
- Handle:
  - `NumberFormatException` if rating is not numeric.
  - `NullPointerException` if rating is null.
- In `Main`, test with valid, null, and invalid inputs.

---

10. Airline Reservation – finally block

Scenario: Booking a flight always requires closing the system.
Requirements:

- Class `Airline`
  - Method: `void bookTicket(String passenger, boolean crash)`
    - If crash = true → throw `RuntimeException("System crash")`.
    - Always print `"System closed"` in `finally`.
- In `Main`, test with normal booking and crash case.