Question 1: Student Management System

Create a simple **Student Management System** with the following requirements:

1. **Student** class with attributes `studentId`, `name`, `age`, and a method to display student details.
2. Create a method to set the student's age, and use exception handling to prevent ages less than 5 or greater than 100.
3. In the main method, create instances of students, attempt to set invalid ages, and catch exceptions for invalid age input using `try-catch`.

Hint: Use `try-catch` to handle `IllegalArgumentException`.

Question 2: Bank Account Transactions

Implement a **Bank Account** class with the following structure:

1. Attributes `accountNumber`, `balance`, and methods `deposit()` and `withdraw()`.
2. In the `withdraw()` method, ensure that a `try-catch` block is used to prevent overdrawing. If an attempt is made to withdraw more than the balance, handle the exception and display an appropriate message.
3. Demonstrate this with a main method where multiple deposit and withdrawal operations are performed.

Hint: Use a `try-catch` block to handle a `ArithmeticException` or `IllegalArgumentException` if an invalid amount is withdrawn.

Question 3: Product Inventory Management

Create a **Product Inventory Management** system with the following requirements:

1. **Product** class with attributes `productId`, `productName`, and `quantity`.
2. Implement a method to update product quantities. If a negative value is added to the inventory, use a `try-catch` block to catch and handle the error, displaying a message saying "Quantity cannot be negative."
3. In the main method, create multiple products and attempt to update quantities with both valid and invalid values.

Hint: Handle the `IllegalArgumentException` if an invalid quantity is entered.

Question 4: Library Book Rental System

Design a **Library Book Rental** system with these requirements:

1. **Book** class with attributes `bookId`, `title`, `isAvailable`, and a method `rentBook()` that sets `isAvailable` to `false` if the book is rented.
2. Implement a `returnBook()` method that sets `isAvailable` to `true`.
3. In `rentBook()`, if the book is already rented, use a `try-catch` block to handle this scenario by printing a message "Book is currently unavailable."
4. Demonstrate this by creating a few books in the main method and attempting to rent already rented books.

Hint: Use `try-catch` to manage the case when the book is already rented.

Question 5: Simple Calculator with Division Operation

Create a **Simple Calculator** class with the following requirements:

1. Methods for basic operations: `add()`, `subtract()`, `multiply()`, and `divide()`.
2. In the `divide()` method, use a `try-catch` block to handle division by zero. If the divisor is zero, catch the `ArithmeticException` and display "Cannot divide by zero."
3. In the main method, perform several calculations, including division by zero, to demonstrate exception handling.

Hint: Handle the `ArithmeticException` when dividing by zero.