Method Overloading Problems

1. **Data Type Conversion Utility**

Problem: Write a class called `DataConverter` that provides various methods to convert different types of data (like integers, floats, strings, etc.) into other types. For example:

- Convert a float to an integer.
- Convert a string to an integer.
- Convert an integer to a string.
- Convert a string to a float.

Use method overloading to handle the different conversions. The class should include methods such as:

- `convert(int)`
- `convert(float)`
- `convert(string)`

Each overloaded method should return the corresponding converted type. Handle exceptions or invalid conversions (e.g., trying to convert a non-numeric string to an integer).

Requirements:

- Use method overloading to define different conversion functions.
- Handle edge cases like invalid conversions.
- Consider both implicit and explicit type conversions.

---

2. **Matrix Operations**

Problem: Implement a class `Matrix` that represents a 2D matrix. Overload the following methods:

- Addition of two matrices.
- Subtraction of two matrices.
- Scalar multiplication of a matrix (by an integer or a float).
- Transpose of a matrix (a method that works on matrices of varying dimensions).

Each operation should be handled through overloaded methods:

- `add(Matrix m)`
- `subtract(Matrix m)`
- `multiply(int scalar)` and `multiply(float scalar)`
- `transpose()`

Ensure that each overloaded method checks for validity (e.g., for matrix dimensions during addition and subtraction) and performs the correct operations.

Requirements:

- Implement the matrix addition, subtraction, and scalar multiplication using method overloading.

- Implement the transpose operation using method overloading.
- Handle matrices of different dimensions.

---

3. **Shape Area Calculator**

Problem: Write a class `Shape` that can calculate the area of different shapes. Overload the method `calculateArea()` to handle the following shapes:

- Rectangle (requires length and breadth).
- Circle (requires radius).
- Triangle (requires base and height).
- Square (requires side length).

Each overloaded `calculateArea` method should take the appropriate parameters and calculate the area of the shape:

- `calculateArea(double length, double breadth)` (for rectangles).
- `calculateArea(double radius)` (for circles).
- `calculateArea(double base, double height)` (for triangles).
- `calculateArea(double side)` (for squares).

Requirements:

- Use method overloading to define different methods for area calculation.
- Each method should calculate the area and return the result.
- Consider edge cases like negative values for dimensions.

---

4. **String Manipulation Library**

Problem: Create a class `StringManipulator` that provides various string operations like reversing, concatenating, and finding substrings. Overload the following methods:

- `reverse()` (to reverse a string).
- `concatenate(string str)` (to concatenate a string with the current string).
- `findSubstring(string substr)` (to find if a substring exists in the current string).
- `substring(int start, int length)` (to extract a substring).

You should overload `reverse()` to handle both an empty string and a string that requires reversing. Similarly, overload `concatenate()` to concatenate different types of strings.

Requirements:

- Use method overloading to define the different string operations.
- Provide meaningful return values such as reversed strings, concatenated strings, or boolean values for substring presence.
- Handle edge cases such as empty strings, invalid start or length in `substring()`, etc.

---

5. **Financial Transaction Logger**

Problem: Design a class `TransactionLogger` that logs transactions in various formats. The class should have an overloaded `logTransaction()` method that logs transactions as:

- An integer value (amount).
- A string value (description of the transaction).
- A float value (amount with decimal precision).

Overload the `logTransaction` method to log the transaction with the type of data passed:

- `logTransaction(int amount)`
- `logTransaction(float amount)`
- `logTransaction(string description)`

Each overloaded method should store the logged transaction in an appropriate format (e.g., saving the amount as a float or integer and the description as a string). You can also overload a `logTransaction` method that accepts both a description and an amount (string and integer/float).

Requirements:

- Use method overloading to allow different types of data to be logged.
- Ensure the transaction is recorded in a readable and meaningful format.
- Handle potential issues, such as logging invalid data or transaction overflow.
