COS 126	Princeton University	Spring 2024

Programming Exam

Before you begin. Read through this page of instructions. Do *not* start the exam (or read the next page) until instructed to do so.

Duration. You have 80 minutes to complete this exam.

Advice. Review the entire exam before starting to write code. Implement the constructor and instance methods in the order given, one at a time, testing in `main()` after you complete each method. The last instance method is the most challenging.

Submission. Submit your solutions on *TigerFile* using the link from the *Exams* page. You may submit multiple times (but only the last version will be graded).

Check Submitted Files. You may click the *Check Submitted Files* button to receive partial feedback on your submission. We will attempt to provide this feature during the exam (but you should not rely upon it).

Grading. Your program will be graded primarily on correctness. Clarity and efficiency will also be considered. You will receive partial credit for a program that implements some of the required functionality. *You will receive a substantial penalty for a program that does not compile.*

Allowed resources. During the exam you may use only the following resources: course textbook; companion booksite; lecture slides; course website; course Ed; your course notes; and your code from the programming assignments or precept. For example, you may not use *Google*, *StackOverflow*, or *ChatGPT*.

No collaboration or communication. Collaboration and communication during this exam are prohibited, except with course staff. A staff member will be outside the exam room to answer clarification question.

No electronic devices or software. Software and computational/communication devices are prohibited, except to the extent needed for taking this exam (such as a laptop, browser, and IntelliJ). For example, you must close all unnecessary applications and browser tabs; disable notifications; and *power off* all other devices (such as cell phones, tablets, smart watches, and earbuds). You *must* use the Princeton wireless network *eduroam*, not a mobile hotspot.

Honor Code pledge. Write and sign the Honor Code pledge by typing the text below in the file `acknowledgments.txt`.

> *I pledge my honor that I will not violate the Honor Code during this examination.*

Electronically sign it by typing `/s/` followed by your name.

After the exam. Discussing or communicating the contents of this exam before solutions have been posted is a violation of the Honor Code, as is accessing *TigerFile*.

Deliverables. For this programming exam, you will submit two files:

1. A Java program `TigerWallet.java`, containing a data type for managing a wallet that contains both cash and a cryptocurrency.
2. An `acknowledgments.txt` file, containing your Honor Code pledge.

TigerWallet. The data type `TigerWallet` represents a wallet that contains a certain amount of cash (in dollars) and a certain amount of a virtual currency (known as *tigercoin*). These two quantities are always non-negative integers. Each wallet is also associated with a particular campus network (such as *Mathey* or *Forbes*).

API. Using the template file `TigerWallet.java` provided in the project folder as a starting point, write a data type that implements the following API:

`public class TigerWallet`	
`public TigerWallet(String network, int cash, int coin)`	*create a new wallet in given network, with specified initial cash and tigercoins*
`public int getCash()`	*amount of cash in this wallet*
`public int getCoin()`	*number of tigercoins in this wallet*
`public String toString()`	*string representation of this wallet*
`public void depositCash(int amount)`	*adds the specified amount of cash to this wallet*
`public void buyCoin(int amount)`	*buys the specified number of tigercoins*
`public void transferCoinTo(int amount, TigerWallet to)`	*transfers the specified number of tigercoins from this wallet to the specified wallet*
`public void mergeWith(TigerWallet[] wallets)`	*merges the specified wallets into this one, transferring all cash and tigercoins*
`public static void main(String[] args)`	*unit tests the* `TigerWallet` *data type*

Here is some additional information about the required behavior:

- The `toString()` method returns a string that represents the wallet in the format (*network*, *cash*, *tigercoins*), such as `"(Mathey, 100, 10)"`.

- The `depositCash()` method increases the amount of cash in the wallet by the deposited amount. (There is no fee for depositing cash.)

- The `buyCoin()` method converts cash to tigercoins within a wallet. Each tigercoin costs $100. There is a $3 fee per transaction. For example, buying 5 tigercoins increases the number of tigercoins in the wallet by 5 and decreases the amount of cash in the wallet by $503.

- The `transferCoinTo()` method transfers tigercoins from one wallet to another. Specifically, the method call `from.transferCoinTo(amount, to)` decreases the number of tigercoins in the `from` wallet (and increases the number of tigercoins in the `to` wallet) by the specified amount. If the two wallets are in different networks, the `from` wallet pays a $3 transaction fee.

- The `mergeWith()` method combines several wallets into one. Specifically, the method call `wallet.mergeWith(wallets)` transfers all of the cash and the tigercoins from the wallets in `wallets[]` to `wallet`, leaving zero cash and tigercoins in the transferred wallets. (There is no fee for merging wallets.)

- *Exceptions.* Throw an `IllegalArgumentException` if the arguments are invalid:
  - Calling `buyCoin()` with insufficient cash to complete the transaction (including the fee).
  - Calling `transferCoinTo()` with
    - ◇ insufficient cash to pay the fee (if applicable), or
    - ◇ insufficient tigercoins to complete the transfer.
  - Calling `wallet.mergeWith(wallets)` if any of the wallets in `wallets[]` are
    - ◇ in a different network than `wallet`, or
    - ◇ if any of the wallets are equal to (i.e., aliases of) one another (or `wallet`).

  For simplicity, you may assume that:

  - The `amount` argument in `depositCash()`, `buyCoin()`, and `transferCoinTo()` is a positive integer.
  - The `cash` and `coin` arguments in the constructor are non-negative integers.
  - No arguments are `null`, including the array elements in `mergeWith()`.

- *Unit testing.* Include a `main()` method that directly calls the constructor and every instance method. You may use this `main()` as a starting point:

```
public static void main(String[] args) {

    // create four wallets
    TigerWallet aja = new TigerWallet("Mathey", 100, 10);
    TigerWallet bob = new TigerWallet("Forbes", 300,  1);
    TigerWallet cai = new TigerWallet("Mathey",   0,  6);
    TigerWallet dee = new TigerWallet("Mathey", 200,  0);

    // print Aja's cash and coin
    StdOut.println("Aja cash: " + aja.getCash());    // Aja cash: 100
    StdOut.println("Aja coin: " + aja.getCoin());    // Aja coin: 10
    StdOut.println();

    // perform some transactions
    aja.depositCash(1000);
    aja.buyCoin(5);
    aja.transferCoinTo(2, cai);

    // print wallets
    StdOut.println("Aja: " + aja);    // Aja: (Mathey, 597, 13)
    StdOut.println("Bob: " + bob);    // Bob: (Forbes, 300, 1)
    StdOut.println("Cai: " + cai);    // Cai: (Mathey, 0, 8)
    StdOut.println("Dee: " + dee);    // Dee: (Mathey, 200, 0)
    StdOut.println();

    // merge Mathey wallets into Aja's wallet
    TigerWallet[] wallets = { cai, dee };
    aja.mergeWith(wallets);

    // print wallets
    StdOut.println("Aja: " + aja);    // Aja: (Mathey, 797, 21)
    StdOut.println("Bob: " + bob);    // Bob: (Forbes, 300, 1)
    StdOut.println("Cai: " + cai);    // Cai: (Mathey, 0, 0)
    StdOut.println("Dee: " + dee);    // Dee: (Mathey, 0, 0)
    StdOut.println();
}
```

Grading. This programming exam has a total of 50 points. Here is the breakdown:

part	*points*
`TigerWallet()`	6
`getCash()`	3
`getCoin()`	3
`toString()`	6
`depositCash()`	6

part	*points*
`buyCoin()`	7
`transferCoinTo()`	8
`mergeWith()`	10
`main()`	1