CSE 1020 Introduction to Computer Science I
A sample final exam

1 (8 marks)

For each of the following pairs of objects, determine whether they are related by aggregation or inheritance. Explain your answers.

(a) *Circle and shape.*

Inheritance, since a circle is-a shape.

(b) *Circle and radius.*

Aggregation, since a circle has-a radius.

(c) *Creditcard and date.*

Aggregation, since a creditcard has-an (expiry)date.

(d) *Creditcard and rewardcard.*

Inheritance, since a rewardcard is-a creditcard.

2 (8 marks)

Which interface from the collections framework (`Set`*,* `List` *or* `Map`*) would be most appropriate for representing each of the collections that are described below. Not only give your choice, but also motivate your choice. Also, include the type parameters (for example, instead of* `Set`*, give its parameterized version, like* `Set<String>`*).*

(a) *The creditcard numbers that have been issued by a creditcard company.*

`Set<Long>`: a collection of `Long`s with no duplicates (each creditcard number is only used once).

(b) *The names of the people who are currently customers of a creditcard company.*

`List<String>`: a collection of `String`s with duplicates allowed (since different people may have the same name).

(c) *The creditcard numbers that have been issued by a creditcard company and, for each issued creditcard number, the corresponding customer name.*

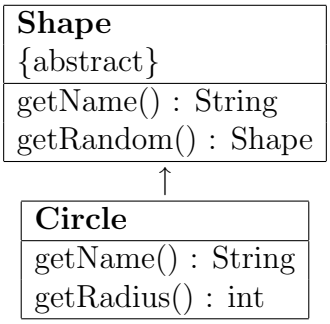
`Map<Long, String>`: mapping each number to the corresponding name.

(d) *The names of the customers of a creditcard company and, for each customer, the creditcard numbers of the card(s) that the customer owns.*

`List<Map<String, Set<Long>>>`: each element of the list maps the name of a customer to the corresponding set of numbers (we need a list since we may have multiple customers with the same name).

3 (6 marks)

Consider the following UML diagram.



Consider the following code snippet (which appears in the body of a main method).

```
Shape shape = Shape.getRandom();
output.print("This shape is a " + shape.getName());
if (Shape instanceof Circle)
{
  output.println(" and its radius is " + ((Circle) shape).getRadius());
}
```

(a) *During early binding, to which method of which class is the method call* `shape.getName()` *associated? Explain your answer.*

The method `getName` of the class `Shape`, since the declared type of `shape` is `Shape`.

(b) *During late binding, to which method of which class is the method call* `shape.getName()` *associated? Explain your answer.*

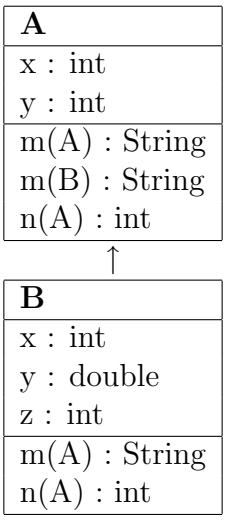
The method `getName` of the class `Circle`, since the actual type of `shape` has to be `Circle` (it cannot be `Shape` because this class is abstract).

(c) *During early binding, to which method of which class is the method call* `shape.getRadius()` *associated? Explain your answer.*

The method `getRadius` of the class `Circle`, since `shape` is casted to `Circle`.

4 (6 marks)

Consider the following UML diagram.



According to the above diagram, class `B` *extends class* `A`.

(a) *How many fields are effectively present in class* `B`*? Explain your answer.*

Three. Both fields of `A` are inherited but are shadowed.

(b) *How many methods are effectively present in class* `B`*? Explain your answer.*

Three: `m(A), m(B) and n(A)`. `m(A)` and `n(A)` of `A` are overwritten. `m(B)` of `A` is inherited.

5 (6 marks)

Consider the following code snippet (which appears in the body of a main method).

```
output.print("Enter a positive integer: ");
int size = input.nextInt();
List<Double> list = new LinkedList<Double>();
for (int i = 0; i < size; i++)
{
  list.add(Math.random());
}
double sum = 0;
for (int i = 0; i < size; i++)
{
```

```
  sum += list.get(i);
}
output.println("Average: " + sum / size);
```

In this fragment, the following methods may throw runtime exceptions.

method name	exception type	condition
`nextInt`	`InputMismatchException`	if the next token is not an integer
`get`	`IndexOutOfBoundsException`	if the index is out of range

Defensive programming and exception-based programming are two approaches to handle these runtime exceptions. For each method that may throw a runtime exception, which approach would be most appropriate? Explain your answers.

For `nextInt`, I would use an exception-based solution since it is simpler than checking if the next token to be read represents an integer.

For `get`, defensive programming is already used: the loop only invokes the method for valid indices.

6 (4 marks)

Consider the following APIs.

public class **Safe**

`public` **Safe**`(Lock lock)`

Construct a safe with the given lock.

Parameters:

`lock` – a lock for the safe.

`public String` **toString**`()`

Return a string representation of this safe. It consists of the string `a safe with` followed by the string representation of the lock of this safe.

Returns:

a string representation of this safe.

public class **Lock**

`public` **Lock**`(int combination)`

Construct a lock with the given combination.

Parameters:

`combination` – a combination for the lock.

`public void` **setCombination**`(int combination)`

Set the combination of this lock to the given combination.

Parameters:

`combination` – the new combination for this lock.

`public String` **toString**`()`

Return a string representation of this lock. It consists of the string `a lock with combination` followed by the combination of this lock.

Returns:

a string representation of this lock.

Describe how one can check if the aggregation of the classes `Safe` *and* `Lock` *is a composition.* If the snippet

```
Lock lock = new Lock(0);
Safe safe = new Safe(lock);
lock.setCombination(1);
output.println(safe);
```

produces as output `a safe with a lock with combination 0`, then it is a composition.