NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
FINAL ASSESSMENT FOR
Semester 2 AY2019/2020
PART 1 of 2

CS2030 Programming Methodology II

May 2020 Time Allowed 45 Minutes

INSTRUCTIONS TO CANDIDATES

1. This assessment is divided into two parts: Part 1 and Part 2.
2. This assessment paper contains 5 questions for Part 1.
3. Write all your answers in the answer boxes provided on Examplify.
4. The total marks for Part 1 is 40. Answer **ALL** questions.
5. This is a **OPEN BOOK** assessment. You are also free to refer to materials online.
6. All questions in this assessment paper use Java 11, unless otherwise specified.

No	Question	Marks
1	Polymorphism	8
2	Stack & Heap	10
3	Assertions	7
4	Exception	10
5	Anonymous Class	5
	Total	**40**

QUESTION 1

Question 1: Polymorphism (8 points)

Consider the following interface `I`:

```
interface I {
  void foo(A a);
  void foo(I i);
}
```

Additionally, there are two classes `A` and `B` such that `B <: A` and both `A` and `B` implement `I`. The class `B` also has an additional method `void foo(B b)`.

There are five different versions of `foo` methods, each containing a single `System.out.print` statement that prints a single digit.

- `foo(A a)` in class `A` prints `1`
- `foo(I i)` in class `A` prints `2`
- `foo(A a)` in class `B` prints `3`
- `foo(I i)` in class `B` prints `4`
- `foo(B b)` in class `B` prints `5`

What is the output of the following code excerpt?

```
A a = new A();
A ab = new B();
B b = new B();
I i = b;
a.foo(a);
b.foo(b);
b.foo(ab);
b.foo(i);
```

Note that this question will be graded by a bot. So, if your answer contains any additional text, such as ”The output is 1234”, ”Line 5 will print 1 because ..” will lead to the answer being marked as wrong even if the intention of the answer is correct.

Solution:

```
1534
```

Question 1: Polymorphism (8 points)

Consider the following interfaces `I1` and `I2`:

```
interface I1 {
  void bar(X x);
  void bar(I1 i);
}

interface I2 {
  void bar(Y y);
}
```

Additionally, there are two classes `X` and `Y` such that `Y <: X`; `X` implements `I1`; and `Y` implements both `I1` and `I2`.

There are five different versions of `bar` methods, each containing a single `System.out.print` statement that prints a single digit.

- `bar(X x)` in class `X` prints `1`
- `bar(I1 i)` in class `X` prints `2`
- `bar(X x)` in class `Y` prints `3`
- `bar(I1 i)` in class `Y` prints `4`
- `bar(Y y)` in class `Y` prints `5`

What is the output of the following code excerpt?

```
X x = new X();
X xy = new Y();
Y y = new Y();
I1 i = new Y();
i.bar(y);
x.bar(y);
y.bar(xy);
x.bar(x);
```

Note that this question will be graded by a bot. So, if your answer contains any additional text, such as ”The output is 1234”, ”Line 5 will print 1 because ..” will lead to the answer being marked as wrong even if the intention of the answer is correct.

Solution:

```
3131
```

Question 1: Polymorphism (8 points)

Consider the following interface `I`:

```
interface I {
  void baz(P p);
  void baz(I i);
}
```

Additionally, there are two classes `P` and `Q` such that `Q <: P` and both `P` and `Q` implement `I`. The class `Q` also has an additional method `void baz(Q q)`.

There are five different versions of `baz` methods, each containing a single `System.out.print` statement that prints a single digit.

- `baz(P p)` in class `P` prints `1`
- `baz(I i)` in class `P` prints `2`
- `baz(P p)` in class `Q` prints `3`
- `baz(I i)` in class `Q` prints `4`
- `baz(Q q)` in class `Q` prints `5`

What is the output of the following code excerpt?

```
P p = new P();
P pq = new Q();
Q q = new Q();
I i = q;
q.baz(q);
q.baz(pq);
p.baz(p);
i.baz(q);
```

Note that this question will be graded by a bot. So, if your answer contains any additional text, such as ”The output is 1234”, ”Line 5 will print 1 because ..” will lead to the answer being marked as wrong even if the intention of the answer is correct.

Solution:

```
5313
```

QUESTION 2

Question 2: Stack and Heap (10 points)

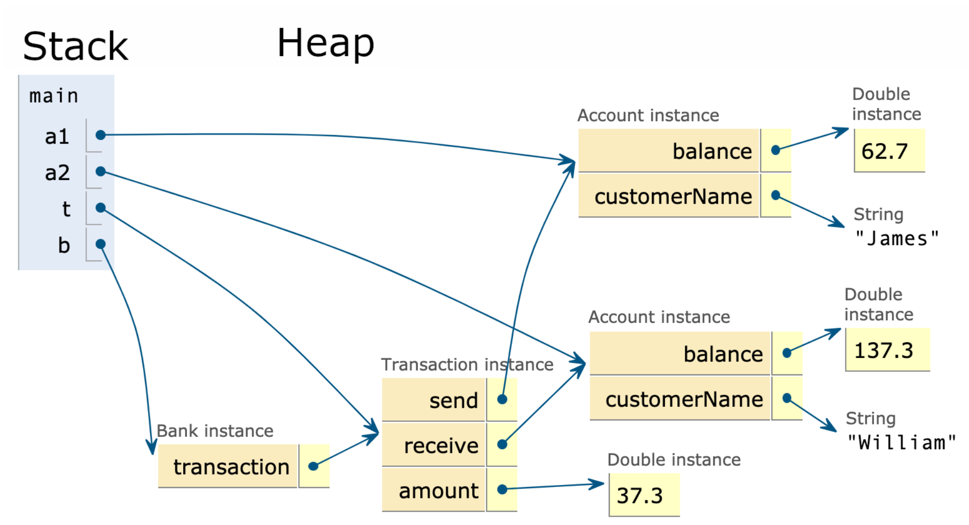
You are trying to maintain and understand an old computer program for managing bank accounts. There are three important classes as described below.

```java
class Bank {
  Transaction transaction;
  void doTransfer(Transaction transaction) {
    this.transaction = transaction;
    this.transaction.complete();
  }
}

class Account {
  Double balance = 100.0;
  String customerName;
  Account(String customerName) {
    this.customerName = customerName;
  }
}

class Transaction {
  Account send;
  Account receive;
  Double amount;
  Transaction(Account send, Account receive, Double amount) {
    this.send = send;
    this.receive = receive;
    this.amount = amount;
  }
  void complete() {
    this.send.balance -= amount;
    this.receive.balance += amount;
  }
}
```

Write the body of the `main` method that will result in the following stack and heap diagram before `main` exits. Note that the local variable `args` is omitted from the figure for clarity and the variables shown in the stack frame of the `main` method are declared accordingly from top to bottom.



Solution:

```
Account a1 = new Account("James");
Account a2 = new Account("William");
Transaction t = new Transaction(a1, a2, 37.3);
Bank b = new Bank();
b.doTransfer(t);
```

Question 2: Stack and Heap (10 points)

You are trying to maintain and understand an old computer program for keeping track of parcel deliveries. There are four important classes as described below.

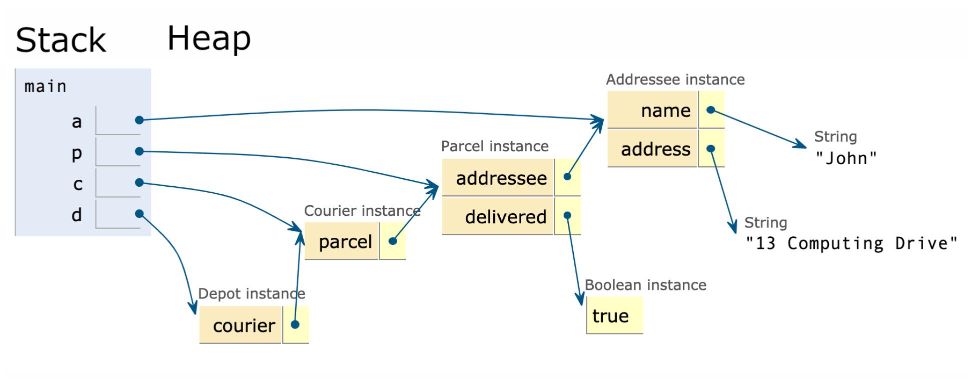
```
class Depot {
  Courier courier;
  void outForDelivery(Courier courier) {
    this.courier = courier;
    this.courier.deliverParcel();
  }
}

class Courier {
  Parcel parcel;
  Courier(Parcel parcel) {
    this.parcel = parcel;
  }
  void deliverParcel() {
    this.parcel.delivered();
  }
}

class Parcel {
  Addressee addressee;
  Boolean delivered = false;
  Parcel(Addressee addressee) {
    this.addressee = addressee;
  }
  void delivered() {
    this.delivered = true;
  }
}

class Addressee {
  String name;
  String address;
  Addressee(String name, String address) {
    this.name = name;
    this.address = address;
  }
}
```

Write a `Main` class and `main` method that will result in the following stack and heap diagram before `main` exits. Note that the local variable `args` is omitted from the figure for clarity and the variables shown in the stack frame of the `main` method are declared accordingly from top to bottom.



Solution:

```
Addressee a = new Addressee("John","13 Computing Drive");
Parcel p = new Parcel(a);
Courier c = new Courier(p);
Depot d = new Depot();
d.outForDelivery(c);
```

Question 2: Stack and Heap (10 points)

You are trying to maintain and understand an old computer program for managing a library. There are four important classes as described below.

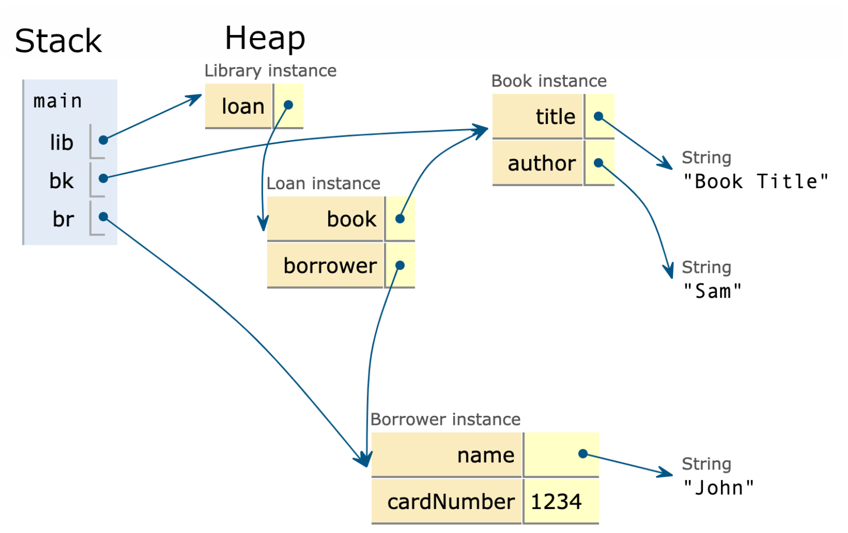
```
class Library {
  Loan loan;
  public void loanBook(Book book, Borrower borrower) {
    loan = new Loan(book, borrower);
  }
}

class Loan {
  Book book;
  Borrower borrower;
  Loan(Book book, Borrower borrower)  {
    this.book = book;
    this.borrower = borrower;
  }
}

class Book {
  String title;
  String author;
  Book(String title, String author) {
    this.title = title;
    this.author = author;
  }
}

class Borrower {
  String name;
  int cardNumber;
  Borrower(String name, int cardNumber) {
    this.name = name;
    this.cardNumber = cardNumber;
  }
}
```

Write a `Main` class and `main` method that will result in the following stack and heap diagram before `main` exits. Note that the local variable `args` is omitted from the figure for clarity and the variables shown in the stack frame of the `main` method are declared accordingly from top to bottom.



Solution:

```
Library lib = new Library();
Book bk = new Book("Book Title","Sam");
Borrower br = new Borrower("John",1234);
lib.loanBook(bk,br);
```

QUESTION 3

Question 3: Assertions (7 points)

The following method `calculateInterest` is used to calculate the interest of a bank account `balance` given a `rate`. The `result` or the maximum amount of interest that can be given is $50 SGD, and the minimum is 0. The balance much be strictly greater than zero, and the rate must be between 0.0 and 1.0 inclusive. Any error here could cost the Bank millions of dollars, so you are tasked with making sure that the method does not get called with incorrect arguments or return an incorrect value.

```
public double calculateInterest(double balance, double rate) {
  // Assertion statement 1 HERE
  ...
  ... Interest Calculation Code
  ...
   // Assertion statement 2 HERE
  return result;
}
```

You are to write two `assert` statements

- One assertion statement that ensures that the input arguments are not outside of the defined ranges.
- One assertion statement that ensures the value to be returned is not outside of the defined range.

Note you do not need to write the method itself, just the assertions.

Solution:

```
assert balance > 0 &&  rate >= 0 &&  rate <= 1.0;
assert result >= 0 &&  result <= 50;
```

Suppose now a subclass is created and it overrides the method above. Write an assert statement for the returned value `result` for the overriding method, *that is different from the assert statements above*, such that the subclass adheres to the Liskov Substitution Principle.

Solution: Any result value that lies between 0 and 50 is ok.

```
assert result > 0 && result < 50;
```

Note that this question will be graded by a bot. Your answer should consists of three Java `assert` statements, one on each line, and nothing else.

Question 3: Assertions (7 points)

A supermarket would like to limit the number items we can buy in a `cart`. The following method `addItems` is used to add `n` number of items to the cart without exceeding 10 items, the maximum number of items in a cart. The `cart` passed to the method should not be empty and the combined size of the `cart` and `n` must not exceed 10. The returned `cart` size also must not exceed 10 items. You are tasked with making sure that the method does not get called with incorrect arguments or return an incorrect value.

```
public List<Item> addItems(List<Item> cart, int n) {
  // Assertion statement 1 HERE
  ...
  ... Add Items Code
  ...
  // Assertion statement 2 HERE
  return newCart;
}
```

You are to write two assert statements

- One assertion statement that ensures that the input arguments are not outside of the defined ranges.
- One assertion statement that ensures the value to be returned is not outside of the defined range.

Note you do not need to write the method itself, just the assertions.

Solution:

```
assert !cart.isEmpty() && cart.size() + n <= 10;
assert newCart.size() <= 10;
```

Suppose now a subclass is created and it overrides the method above. Write an assert statement for the returned value `newCart` for the overriding method, *that is different from the assert statements above*, such that the subclass adheres to the Liskov Substitution Principle.

Solution: Any returned value that is stricter is ok.

```
newCart.count() < 10;
```

Note that this question will be graded by a bot. Your answer should consists of three Java `assert` statements, one on each line, and nothing else.

Question 3: Assertions (7 points)

An airline has a problem in their computer system and has been double-booking many flights. The following method `buyTicket` is used to buy a seat on a flight. The `seatList` contains all passengers who have already bought a seat on the flight. The `seatList` passed to the method should have at least one Passenger already and should not contain the passed passenger `p`. The returned `seatList` size also must not exceed the size of the plane (250) and should contain `p`. You are tasked with making sure that the method does not get called with incorrect arguments or return an incorrect value.

```
public List<Passenger> buyTicket(List<Passenger> seatList, Passenger p) {
  // Assertion statement 1 HERE
  ...
  ... Add
  ...
  // Assertion statement 2 HERE
  return seatList;
}
```

You are to write two `assert` statements

- One assertion statement that ensures that the input arguments are not outside of the defined ranges.
- One assertion statement that ensures the value to be returned is not outside of the defined range.

Note you do not need to write the method itself, just the assertions.

Solution:

```
assert seatList.size() > 0 && !seatList.contains(p);
assert seatList.size() <= 250  && seatList.contains(p);
```

Suppose now a subclass is created and it overrides the method above. Write an assert statement for the returned value `seatList` for the overriding method, *that is different from the assert statements above*, such that the subclass adheres to the Liskov Substitution Principle.

Solution: Any result value that is stricter is ok. For instance,

```
assert seatList.count() < 250 && seatList.contains(p);
```

Note that this question will be graded by a bot. Your answer should consists of three Java `assert` statements, one on each line, and nothing else.

QUESTION 4

Question 4: Exceptions (10 points)

Consider the following hierarchy of Exceptions:

```
class FileIOException extends Exception { }

class FileNotFoundException extends FileIOException { }

class FileMetadataException extends FileIOException { }

class IncorrectPermissionException extends FileMetadataException { }

class IncorrectSizeException extends FileMetadataException { }

class Main {
  public static void loadFile() throws FileIOException {
    ...
  }
  public static void main(String[] args) {
    // Your code will be put here
  }
}
```

`Main.loadFile()` can throw any of the above exceptions.

Write code to run `Main.loadFile()` and then catch each exception individually and print out the name of the exception. Do not write the method signature of `main`, just write the method body.

Solution:

```
try {
  Main.loadFile();
}
catch (IncorrectPermissionException e) {
  System.out.println("IncorrectPermissionException");
}
catch (IncorrectSizeException e) {
  System.out.println("IncorrectSizeException");
}
catch (FileMetadataException e) {
  System.out.println("FileMetadataException");
}
catch (FileNotFoundException e) {
  System.out.println("FileNotFoundException");
}
catch (FileIOException e) {
  System.out.println("FileIOException");
}
```

Question 4: Exceptions (10 points)

Consider the following hierarchy of Exceptions:

```java
class DatabaseException extends Exception { }

class TableException extends DatabaseException { }

class ReadException extends TableException { }

class ElementNotFoundException extends ReadException { }

class WriteException extends TableException { }

class NoPermissionException extends WriteException { }

class Main {
  public static void accessDatabase() throws DatabaseException {
    ...
  }
  public static void main(String[] args) {
    // Your code will be put here
  }
}
```

`Main.accessDatabase()` can throw any of the above exceptions.

Write code to run `Main.accessDatabase()` and then catch each exception individually and print out the name of the exception. Do not write the method signature of `main`, just write the method body.

Solution:

```java
try {
  Main.accessDatabase();
}
catch (NoPermissionException e) {
  System.out.println("NoPermissionException");
}
catch (WriteException e) {
  System.out.println("WriteException");
}
catch (ElementNotFoundException e) {
  System.out.println("ElementNotFoundException");
}
catch (ReadException e) {
  System.out.println("ReadException");
}
catch (TableException e) {
  System.out.println("TableException");
}
```

```
catch (DatabaseException e) {
  System.out.println("DatabaseException");
}
```

Question 4: Exceptions (10 points)

Consider the following hierarchy of Exceptions:

```
class InternetException extends Exception { }

class ConnectionFailureException extends InternetException { }

class AuthenticationException extends ConnectionFailureException { }

class LoginFailureException extends AuthenticationException { }

class NoSuchUsernameException extends LoginFailureException { }

class PasswordIncorrectException extends LoginFailureException { }

class Main {
  public static void login() throws InternetException {
    ...
  }

  public static void main(String[] args) {
    // Your code will be put here
  }
}
```

`Main.login()` can throw any of the above exceptions.

Write code to run `Main.login()` and then catch each exception individually and print out the name of the exception. Do not write the method signature of `main`, just write the method body.

Solution:

```
try {
  Main.login();
}
catch (PasswordIncorrectException e) {
  System.out.println("PasswordIncorrectException");
}
catch (NoSuchUsernameException e) {
  System.out.println("NoSuchUsernameException");
}
catch (LoginFailureException e) {
  System.out.println("LoginFailureException");
}
catch (AuthenticationException e) {
  System.out.println("AuthenticationException");
}
catch (ConnectionFailureException e) {
  System.out.println("ConnectionFailureException");
}
```

```
catch (InternetException e) {
  System.out.println("InternetException");
}
```

QUESTION 5

Question 5: Anonymous Class (5 points)

Consider the following lambda expression:

```
BiFunction<Character,String,Integer> s;
s = (i, j) -> (i + j).length();
```

Write the equivalent statement to intialize s using an anonymous class.

Solution:

```
s = new BiFunction<Character, String, Integer>() {
    @Override
    public Integer apply(Character i, String j) {
      return (i + j).length();
    }
};
```

Question 5: Anonymous Class (5 points)

Consider the following lambda expression:

```
BiFunction<Double, Float, Long> r;
r = (a, b) -> Math.round(a + b);
```

Write the equivalent statement to initialize `r` using an anonymous class.

Solution:

```
r = new BiFunction<Double, Float, Long>() {
    @Override
    public Long apply(Double a, Float b) {
        return Math.round(a + b);
    }
};
```

Question 5: Anonymous Class (5 points)

Consider the following lambda expression:

```
BiFunction<String, Integer, String> c;
c = (x, y) -> String.format("%s:%d", x, y);
```

Write the equivalent statement to intialize c using an anonymous class.

Solution:

```
c = new BiFunction<String,Integer,String>() {
    @Override
    public String apply(String x, Integer y) {
      return String.format("%s:%d",x,y);
    }
};
```

The End