Notes and Instructions:

1. Create a folder in your computer device with your full name (FirstName_LastName) and save all your work in it.
2. Download and save the java files ***TelecomCustomers*** and ***Node*** that were implemented and the text files ***Service1*** and ***Service2*** in your created folder.
3. Use the same variable names in the UML and description, and meaningful names for other variables.
4. You **MUST** make use of existing methods when appropriate.
5. To submit your work, **compress** your files in a zip folder and upload it to LMS.
6. If you encounter any problem to upload your work, send it to your lab instructor email as shown below:

Instructor' name	**Email**
Alaa Al Ali	alaalali@KSU.EDU.SA
Arwa Alrubaian	aalrubaian@KSU.EDU.SA
Faten Al Omar	Teacher.faten.cs@gmail.com
Heba Khojah	Heba_khojah@hotmail.com
Eman Al bilali	ealbilali@KSU.EDU.SA
Malak Al mojaly	malmojaly@KSU.EDU.SA
Norah Al fantoukh	nfantoukh@KSU.EDU.SA
Rawabi Al waneen	ralwaneen@KSU.EDU.SA

Question:

1) **The UML is given on page 5,** the classes *Node* and *TelecomCustomers* have already been implemented for you**; you need to use them properly.**
2) Implement all the other classes and interface that are shown in the UML with their descriptions in each class.
3) In addition, add and implement the methods ***economicalCustomersBills*** and ***saveCustomesToFile*** in class ***TelecomCustomers.***
4) You may use setters and getters when necessary, otherwise, you are **NOT** allowed to add any extra method or attribute not required in the UML.
5) Define a new checked exception of type ***InvalidCategoryException***

Classes description:

1. Class ***Service.***

Attributes:

- ***sId***: The service id that has the following format: for **mobile** service **M##** and for **internet** service: **I##. (Note: #** is a digit).
- ***plan:*** The service plan could be either "***postpaid***" or "***prepaid***".
- ***tariff:*** The monthly package fee of a service.

Methods:

- ***Service (id:** String, **p:** String, **tariff:** String**):*** It is a constructor to initialize data**.**
- ***issueBill():*** It is an abstract method.
- ***toString():*** It returns a string of all service information.

2. Class ***Mobile:***

Attributes:

- ***mNum:*** The mobile number.
- ***minutesCalls:*** The total number of minutes of the mobile's calls.
- ***netUsage*:** The total number of internet data usage in *GB*.

Methods:

- ***Mobile (id:** String**, plan:** String**, tariff:** double **, num:**String **, min:**int **, net:** double **):*** constructor to initialize data.
- ***Mobile (m:** Mobile**):*** copy constructor.
- ***issueBill( ):*** It returns the value of the mobile service bill according to the following formula:
    - if the mobile service plan is "***postpaid***" the bill will be calculated as:

      tariff + (minutesCalls* 0.15) + (netUsage * 0.05)

    - if the mobile service plan is "***prepaid***" the bill will be calculated as:

      tariff + (minutesCalls * 0.10) + (netUsage * 0.05)
- ***toString( ):*** It returns a string of all information of the mobile service.

3. Class ***Internet:***

Attributes:

- ***speed:*** The speed of the internet in *Mbps*.
- ***dataUsage:*** The total number of internet data usage in *GB*.

Methods:

- ***Internet (id:** String**, plan:** String**, fee:**double**, s:**double**, data:**double **)*** constructor to initialize data.
- ***Internet (I:** Internet**):*** copy constructor.

- ***issueBill( ):*** It returns the value of the internet service bill according to the following formula: tariff * (dataUsage / speed) * 0.25.

  <u>Note</u>: the formula above is applicable for **postpaid** and **prepaid** internet services.
- ***toString( ):*** It returns a string of all information of the internet service.

4. Class ***<u>Customer:</u>***

<u>Attributes:</u>

- ***cID:*** The customer id.
- ***cName:*** The customer name.
- ***cCategory:*** The customer category could be, either "***standard***" or "***Gold***"
- ***numServices:*** The **actual** number of services that are provided to the customer.

<u>Methods:</u>

- ***Customer (cID:*** *int*, ***cName:*** *String*, ***cCategory:*** *String*, ***size:*** *int* ***):*** It initializes the customer information.
  - ***size:*** the maximum number of **possible** services that customer has.
  - If the ***cCategory*** is neither ***"standard"*** nor ***"Gold"***, an ***InvalidCategoryException*** should be generated and thrown with an appropriate message to the calling environment.
- ***addServiceFromfile (fname:*** *String*)***:*** It <u>loads</u> all information of services whether they are **mobile services** or **internet services** that are provided to the customer <u>from</u> **a given text file *fname.*** The information of each service is represented in a single line in the text file ordered as following:
  - **service id, plan, tariff, mobile num, minutes calls** and **net usage** for a mobile service.

    <u>Note:</u> the format of **mobile service id** is: **"M##".** e.g **M**45
  - **service id, plan, tariff, internet speed,** and **data usage** for an internet service.

    <u>Note:</u> the format of **internet service id** is **"I##".** e.g **I**23

Then, it <u>adds</u> each read service to the list of services that are provided to the customer.

<u>Hint:</u> Notice that the list of services may not be large enough to include all services from the file. Therefore, **the method should handle an appropriate exception for this case with displaying a meaningful message.**

- ***lowestmServiceBill (plan:*** *String* ***):*** it returns the <u>lowest</u> **mobile** service **bill** with a given plan for a customer.

  <u>Note:</u> the method should not change the content or the order of the list.
- ***toString():*** It returns a string of all customer's information including all services of the customer.

5. Class *<u>TelecomCustomers</u>*

<u>Methods:</u>

- ***economicalCustomersBills (plan:*** *String*, ***rate:*** *double* ***):*** It returns a new list of customers who have the mobile bill, with a given **<u>plan,</u>** is **less** than or **equal** to a given bill **<u>rate</u>**. i.e, If

the lowest the issued bill for each customer with a given plan is less than or equal the given rate, add it to the new list of customers.
Note: the method should not change the content or the order of the list.

6. **Interface *IntputOutputFile***

Methods:

- ***saveCustomersToFile (filename:** String **):*** It saves a list of customers objects with the list of services to the object file ***fName.*** The saving is done by copying objects of the list into a file ***object by object.*** Each time an object is copied to the file, this object is removed from the customers list.

7. **Class *Test:***

Create a new Application class named ***Test*** with a main method to perform the following:

a. Create a **queue** of customers using the class ***TelecomCustomers*** named ***cList***.
b. Add two customers to ***cList*** with following information:

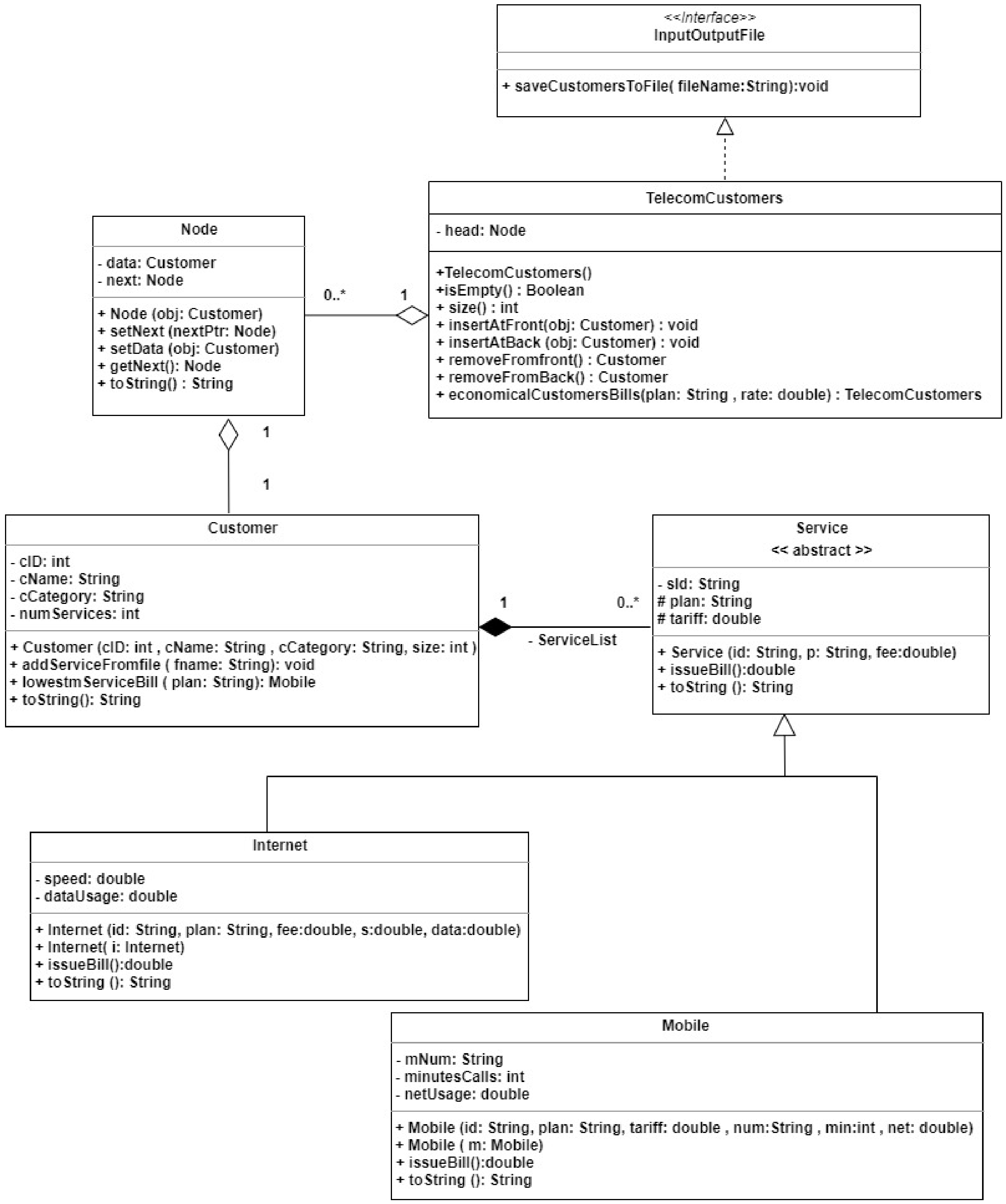
Customer	Id	name	category	# of services
1	101	Ali	*Input from the user	6
2	102	Maha	*Input from the user	7

*Prompt the user to input the customer category that could either be, "**Gold**" or "**standard**". If the input value is invalid, **allow the user to re-enter until a correct value is read.**

c. Add the list of services for each customer by loading them from the txt files. The list of services for customer1 are listed in the text file named ***"Service1.txt"*** and for customer2 are listed in the file ***"Service2.txt"***.
d. Create **a new** queue of customers named ***ecoList*** , for the customers with the ***rate*** of mobile bill ***120*** and ***postpaid*** mobile services ***plan.***
e. Display all the information of the customers in ***ecoList*** with all services.
Note: do not change the content or the order of the list.
f. Save each customer information in ***cList*** queue into the object file ***"ServedCustomers.dat".***

Note: Your main method should handle all possible exceptions by displaying a meaningful message.

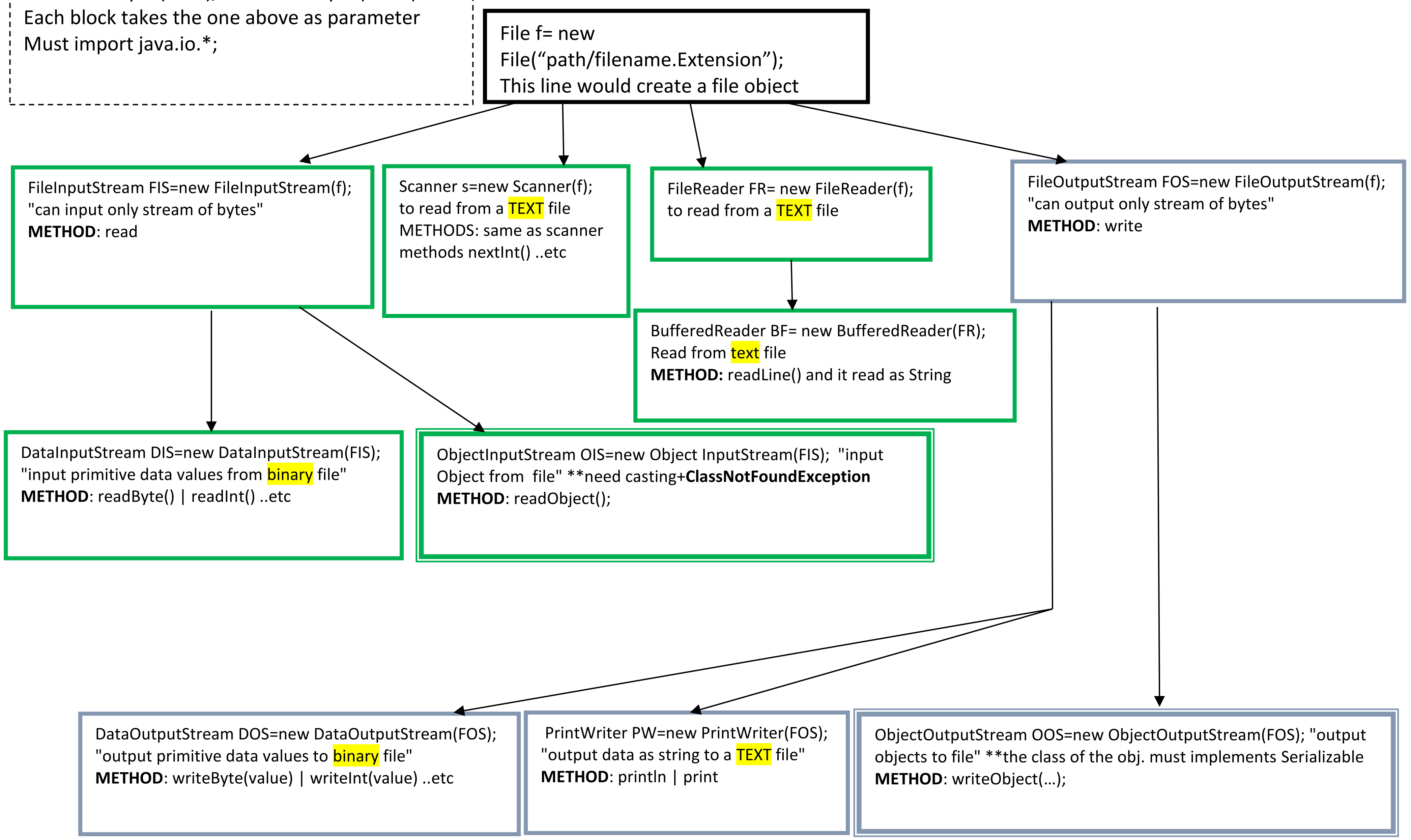
UML:



<<Interface>>
InputOutputFile
+ saveCustomersToFile(fileName:String):void
TelecomCustomers
- head: Node
+TelecomCustomers()
+isEmpty() : Boolean
+ size() : int
+ insertAtFront(obj: Customer) : void
+ insertAtBack (obj: Customer) : void
+ removeFromfront() : Customer
+ removeFromBack() : Customer
+ economicalCustomersBills(plan: String , rate: double) : TelecomCustomers
Node
- data: Customer
- next: Node
+ Node (obj: Customer)
+ setNext (nextPtr: Node)
+ setData (obj: Customer)
+ getNext(): Node
+ toString() : String
0..*
1
1
1
Customer
- cID: int
- cName: String
- cCategory: String
- numServices: int
+ Customer (cID: int , cName: String , cCategory: String, size: int)
+ addServiceFromfile (fname: String): void
+ lowestmServiceBill (plan: String): Mobile
+ toString(): String
1
0..*
- ServiceList
Service
<< abstract >>
- sId: String
# plan: String
# tariff: double
+ Service (id: String, p: String, fee:double)
+ issueBill():double
+ toString (): String
Internet
- speed: double
- dataUsage: double
+ Internet (id: String, plan: String, fee:double, s:double, data:double)
+ Internet(i: Internet)
+ issueBill():double
+ toString (): String
Mobile
- mNum: String
- minutesCalls: int
- netUsage: double
+ Mobile (id: String, plan: String, tariff: double , num:String , min:int , net: double)
+ Mobile (m: Mobile)
+ issueBill():double
+ toString (): String

IO Files and Streams

Green for input(read)/ blue for output(write).
Each block takes the one above as parameter
Must import java.io.*;



File f= new File("path/filename.Extension");
This line would create a file object

FileInputStream FIS=new FileInputStream(f);
"can input only stream of bytes"
METHOD: read

Scanner s=new Scanner(f);
to read from a TEXT file
METHODS: same as scanner methods nextInt() ..etc

FileReader FR= new FileReader(f);
to read from a TEXT file

FileOutputStream FOS=new FileOutputStream(f);
"can output only stream of bytes"
METHOD: write

BufferedReader BF= new BufferedReader(FR);
Read from text file
METHOD: readLine() and it read as String

DataInputStream DIS=new DataInputStream(FIS);
"input primitive data values from binary file"
METHOD: readByte() | readInt() ..etc

ObjectInputStream OIS=new Object InputStream(FIS); "input Object from file" **need casting+**ClassNotFoundException**
METHOD: readObject();

DataOutputStream DOS=new DataOutputStream(FOS);
"output primitive data values to binary file"
METHOD: writeByte(value) | writeInt(value) ..etc

PrintWriter PW=new PrintWriter(FOS);
"output data as string to a TEXT file"
METHOD: println | print

ObjectOutputStream OOS=new ObjectOutputStream(FOS); "output objects to file" **the class of the obj. must implements Serializable
METHOD: writeObject(...);

Exception Hierarchy:

```
Exception
   — ClassNotFoundException
   — CloneNotSupportedException
   — InstantiationException
   — NoSuchFileException
   — NoSuchMethodException
   — RuntimeException
           — ArithmeticException
           — ClassCastException
           — IllegalArguemtException
                     — NumberFormatException
           — IndexOutOfBoundsException
                     — ArrayIndexOutOfBoundsException
                     — StringIndexOutOfBoundsException
           — NullPointerException
           — EmptyStackException
           — NoSuchElementException
                     — InputMismatchException
   — TooManyListenersException
   — IOException
           — CharConversionException
           — EOFException
           — FileNotFoundException
           — InterruptedException
           — ObjectStreamException
                     — InvalidClassException
                     — InvalidObjectException
                     — NotActiveException
                     — StreamCorruptedException
                     — WriteAbortedException
```