**PSG COLLEGE OF TECHNOLOGY, COIMBATORE**

**DEPARTMENT OF APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCES JAVA PROGRAMMING LAB**

**PROBLEM SHEET – Exceptions**

**ArrayIndexOutOfBoundsException**

ArrayIndexOutOfBoundsException, occurs when the program tries to access the array beyond its size. As we know arrays have fixed size. So when you try to use array beyond its size it throws this exception. Let's try to handle this exception.

Handling this exception will also prove to be good for our application. F or example, if there are only 100 seats in the event and the user tries to book the 105th seat, it will throw this exception. So you must handle it to do a specific job.

Create an array of size 100 and assume it as seat array. Get the tickets to be booked from the user and handle any exception that occurs in **Main** Class. At last display all the tickets booked.

**Input and Output format:**

The first line of input consists of an integer which corresponds to the number of seats to be booked. The next n lines of input consist of the integer which corresponds to the seat number. Refer to sample Input and Output for formatting specifications.

**Note: All Texts in bold corresponds to the input and rest are output. Sample Input and Output 1:**

Enter the number of seats to be booked:

**5**

Enter the seat number 1

**23**

Enter the seat number 2

**42**

Enter the seat number 3

**65**

Enter the seat number 4

**81**

Enter the seat number 5

**100**

The seats booked are:

23

42

65

81

100

**Sample Input and Output 2:**

Enter the number of seats to be booked:

**4**

Enter the seat number 1

**12**

Enter the seat number 2

**101**

java.lang.ArrayIndexOutOfBoundsException: 100

| import java.lang.\*;  import java.util.\*;  public class Main {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int[] seats = new int[100];  System.out.println("Enter the number of seats to be booked :");  int total = Integer.parseInt(sc.nextLine());  for (int i = 0; i < total; i++) {  try {  System.out.println("Enter seat no. " + (i + 1) + ": ");  seats[Integer.parseInt(sc.nextLine())] = 1;  System.out.println("The seat was booked.");  } catch (ArrayIndexOutOfBoundsException e) {  System.err.println(e);  }  }  }  } |
| --- |
|  |

**SeatNotAvailableException**

An organization is organizing a charity fate for the well being of poor kids. Since the manager was running short on time, he asked you to help him with the ticket bookings. You being from a programming background decide to design a program that asks the user about the seat number they want. Seat booking details are stored in an array. If the seat number requested is available booking should be done else print the message " **SeatNotAvailableException**". If the seat number requested is not in the range throws an exception **ArrayIndexOutOfBoundsException.**

Create a class **SeatNotAvailableException** that extends Exception.

Create an array of size n\*n (n rows each with n seats) which is got from the user. Get the tickets to be booked from the user and handle any exception that occurs in **Main** Class. (Take seat numbers from 0 to (n\*n)-1)

**Note**: Vacant seats are denoted by (**0)** and booked seats by (**1)**. Show message as "**Already Booked**" as a Custom exception.

**Input and Output format:**

Refer sample Input and Output for formatting specifications.t of the output.

**[All Texts in bold corresponds to the input and rest are output]**

**Sample Input and Output 1:**

Enter the number of rows and columns of the show:

**3**

Enter the number of seats to be booked:

**2**

Enter the seat number 1

**8**

Enter the seat number 2

The seats booked are:

1 0 0

0 0 0

0 0 1

**Sample Input and Output 2:**

Enter the number of rows and columns of the show:

**3**

Enter the number of seats to be booked:

**2**

Enter the seat number 1

**9**

java.lang.ArrayIndexOutOfBoundsException: 9

The seats booked are:

0 0 0

0 0 0

0 0 0

**Sample Input and Output 3:**

Enter the number of rows and columns of the show:

**4**

Enter the number of seats to be booked:

**3**

Enter the seat number 1

**15**

Enter the seat number 2

**14**

Enter the seat number 3

**15**

SeatNotAvailableException: Already Booked

The seats booked are:

0 0 0 0

0 0 0 0

0 0 0 0

0 0 1 1

|  |
| --- |

**Duplicate mobile number exception**

Write a java program to find the duplicate mobile number using the exception handling mechanism.

**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**

Create a Class called **ContactDetail** with the following private attributes.

| **Attributes** | **Datatype** |
| --- | --- |
| mobile | String |
| alternateMobile | String |
| landLine | String |
| email | String |
| address | String |

Include getters and setters.

Include default and parameterized constructors.

Format for a parameterized constructor is **ContactDetail(String mobile, String alternateMobile,String landLine, String email, String address)**

Override the **toString()** method to display the Contact details as specified.

Create a class called **ContactDetailBO** with following methods

| **Method** | **Description** |
| --- | --- |
| static void validate(String mobile,String alternateMobile) | This method throws DuplicateMobileNumber exception  if the mobile and alternateMobile are the same. |

| import java.lang.\*;  import java.util.\*;  public class Main {  public static void main(String[] args) {  try {  ContactDetail cd1 = new ContactDetail("99942827637","99042827627","ll1","mail1", "a1");  ContactDetailBO.validate(cd1.getMobile(), cd1.getAlternateMobile());  } catch (DuplicateMobileNumberException e) {  System.err.println(e);  }  // Exception thrown in this case  try {  ContactDetail cd2 = new ContactDetail("99942827637","99942827637","ll2","mail2", "a2");  ContactDetailBO.validate(cd2.getMobile(), cd2.getAlternateMobile());  } catch (DuplicateMobileNumberException e) {  System.err.println(e);  }  }  }  class ContactDetail {  private String mobile;  private String alternateMobile;  private String landline;  private String email;  private String address;  public ContactDetail() {  }  public ContactDetail(String mobile, String alternateMobile, String landline, String email, String address) {  this.mobile = mobile;  this.alternateMobile = alternateMobile;  this.landline = landline;  this.email = email;  this.address = address;  }  public String getMobile() {  return mobile;  }  public void setMobile(String mobile) {  this.mobile = mobile;  }  public String getAlternateMobile() {  return alternateMobile;  }  public void setAlternateMobile(String alternateMobile) {  this.alternateMobile = alternateMobile;  }  public String getLandline() {  return landline;  }  public void setLandline(String landline) {  this.landline = landline;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  public String getAddress() {  return address;  }  public void setAddress(String address) {  this.address = address;  }  @Override  public String toString() {  return "\nContactDetail{" +  "\n\tmobile='" + mobile + '\'' +  ",\n\talternateMobile='" + alternateMobile + '\'' +  ",\n\tlandline='" + landline + '\'' +  ",\n\temail='" + email + '\'' +  ",\n\taddress='" + address + '\'' +  "\n}";  }  }  class ContactDetailBO{  static void validate(String mobile, String alternateMobile) throws DuplicateMobileNumberException {  if(mobile == alternateMobile){  throw new DuplicateMobileNumberException("The mobile number and the alternate mobile number are the same => " + mobile + ".");  }  else return;  }  }  class DuplicateMobileNumberException extends Exception {  public DuplicateMobileNumberException(String message) {  super(message);  }  } |
| --- |

Create a driver class called **Main**. In the Main method, obtain inputs from the user. Validate the mobile and alternateMobile and display the ContactDetail if no exception occurs else handle the exception.

Pass the exception message as "**Mobile number and alternate mobile number are same**". If mobile and alternateMobile are the same.

**Input and Output format:**

Refer to sample Input and Output for formatting specifications.

**Note: All text in bold corresponds to the input and rest corresponds to the output. Sample Input and Output 1:**

Enter the contact details

**9874563210,9874563210,0447896541,johndoe@abc.in,22nd street kk nagar chennai** DuplicateMobileNumberException: Mobile number and alternate mobile number are same

**Sample Input and Output 2:**

Enter the contact details

**9874563210,9876543211,0447896541,johndoe@abc.in,22nd lane RR nagar kovai** Mobile:9874563210

Alternate mobile:9876543211

LandLine:0447896541

Email:johndoe@abc.in

Address:22nd lane RR nagar kovai

**EventTypeDoesNotExistsException**

Now we know how to create a custom exception that suits our needs. So let's try it in a part of our application. All the events should be of the predefined event type. So hardcode some event types before and check if the event entered having the same id or throw

EventTypeDoesNotExistsException, and ask the user to reenter the event typeId until they got it right.

Create a class **Event** with the following private attributes,

| **Attribute** | **Datatype** |
| --- | --- |
| name | String |
| detail | String |
| ownerName | String |
| typeId | Long |

Create a class **EventType** with following private attributes,

| **Attribute** | **Data type** |
| --- | --- |
| id | Long |
| name | String |

Include appropriate getters/setters, default and parameterized constructors and override toString() to display the details.

Hard code few event types in driver class **Main**- (Stage event with id-1),(Exhibition with id-2),(Sports meet with id-3).

Create the following static method in the Main class,

| **Method** | **Description** |
| --- | --- |
| public static Boolean isValid(Long  typeId,List<EventType> typeList) | This method checks the validity of the event by comparing the type id with the available types. If valid return true, else throw custom Exception. |

Create the custom exception class **EventTypeDoesNotExistsException** that extends Exception.

**Input and Output format:**

CSV format of the input is **(name,detail,ownerName,typeId)**.

Use "**%-15s%-15s%-15s%-15s**" to print in tabular form.

Show message as "**No event type available with the given id**" in Custom exception. Refer to sample Input and Output for formatting specifications.

**Note: All Texts in bold corresponds to the input and rest are output.**

**Sample Input and Output 1:**

Enter the number of the events:

**2**

Enter the details of event 1

**Book Fair,20% discount,John,2**

Enter the details of event 2

**Marathon,50km race,Jane,3**

The events entered are:

Name Details Owner name Eventtypeid

Book Fair 20% discount John 2

Marathon 50km race Jane 3

**Sample Input and Output 2:**

Enter the number of the events:

**2**

Enter the details of event 1

**Book Fair,20% discount,John,2**

Enter the details of event 2

**Marathon,50km race,Jane,5**

EventTypeDoesNotExistsException: No event type available with the given id Enter the correct event type id:

**4**

EventTypeDoesNotExistsException: No event type available with the given id Enter the correct event type id:

**3**

The events entered are:

Name Details Owner name Eventtypeid

Book Fair 20% discount John 2

Marathon 50km race Jane 3

**Weak password Exception**

A typical requirement of a custom exception would be for validation purposes. In this exercise, Let's validate a password input. A password is said to be strong if it satisfies the following criteria i) It should be a minimum of 10 characters and a maximum of 20 characters. ii) It should contain at least one digit.

iii)It should contain at least one special character (non-numeric, non-alphabetic). iv)It should contain at least one letter.

If the password fails any one of the criteria, it is considered as weak.

**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**

Create a class called **User** with the following private attributes.

| **Attributes** | **Datatype** |
| --- | --- |
| name | String |
| mobile | String |
| username | String |

| password | String |
| --- | --- |

Include getters and setters.

Include default and parameterized constructors.

Format for the parameterized constructor is **User(String name, String mobile, String username, String password)**

Override the **toString()** method to display the User detail

Create a class called **UserBO** with the following methods.

| **Method** | **Description** |
| --- | --- |
| static void validate(User u) | This method throws WeakPasswordNumber exception if the Password is weak. |

Create a driver class called **Main**. In the Main method, obtain inputs from the user. Validate the password and if there is an exception, handle the exception.

Pass the exception message as "**Your password is weak**".

**Sample Input and Output:**

Refer to sample Input and Output for formatting specifications.

**Note: All text in bold corresponds to the input and rest corresponds to the output. Sample Input and Output 1:**

Enter the user details

**John Doe,9876543210,john,johndoe**

WeakPasswordException: Your password is weak

**Sample Input and Output 2:**

Enter the user details

**Jane doe,9876543210,Jane,Janedoe@123**

Name:Jane doe

Mobile:9876543210

Username:Jane

Password:Janedoe@123

**HallNotAvailableException**

Create a custom exception **HallNotAvailableException** which checks the availability of the hall and throws an exception if the user tries to book the hall for different events on the same day.

Write a java program to find the availability of the hall using the exception handling mechanism. Create a class **Hall** with the following private attributes,

| **Attribute** | **Data Type** |
| --- | --- |
| name | String |
| contactNumber | String |
| costPerDay | Double |
| ownerName | String |

Create a class **Event** with the following private attributes,

| **Attribute** | **Data type** |
| --- | --- |
| name | String |
| detail | String |
| ownerName | String |
| type | String |

Create a bridge class **HallBooking** that bridges Hall and Event with following attributes,

| **Attribute** | **Data type** |
| --- | --- |
| hall | Hall |
| event | Event |
| eventDate | Date |
| price | Double |

Include appropriate getters/setters, default and parameterized constructors.

Override **toString()** to display the details as specified.

Create a class **HallBookingBO** to do the manipulations of HallBooking class with following methods,

| **Method** | **Description** |
| --- | --- |
| public static Boolean  validateHallBooking(ArrayList<HallBooking>bookingList,HallBooking booking) | throws Exception if the same hall is booked on the same date  for different events else return true |

| import java.lang.\*;  import java.util.\*;  import com.sun.jdi.ArrayReference;  public class Main {  public static void main(String[] args) {  ArrayList<HallBooking> hbList = new ArrayList<>();  Hall h1 = new Hall("h1", "c1", 10.00, "o1");  Hall h2 = new Hall("h2", "c2", 20.00, "o2");  Event6 e1 = new Event6("e1", "d1", "ow1", "t1");  Event6 e2 = new Event6("e2", "d2", "ow2", "t2");  Event6 e3 = new Event6("e3", "d3", "ow3", "t3");  Event6 e4 = new Event6("e4", "d4", "ow4", "t4");  HallBooking hb1 = new HallBooking(h1, e1, new Date(1,1,1), 10.00);  HallBooking hb2 = new HallBooking(h2, e1, new Date(3,3,3), 30.00);  HallBooking hb3 = new HallBooking(h2, e4, new Date(4,4,4), 40.00);  hbList.add(hb1);  hbList.add(hb2);  hbList.add(hb3);  HallBooking hbNew = new HallBooking(h1, e2, new Date(1,1,1), 20.00);  try {  if(HallBookingBO.validateHallBooking(hbList, hbNew)) {  hbList.add(hbNew);  }  } catch (HallNotAvailableException e) {  System.err.println(e);  }  }  }  class Hall {  private String name;  private String contactNumber;  private double costPerDay;  private String ownerName;  public Hall(String name, String contactNumber, double costPerDay, String ownerName) {  this.name = name;  this.contactNumber = contactNumber;  this.costPerDay = costPerDay;  this.ownerName = ownerName;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public String getContactNumber() {  return contactNumber;  }  public void setContactNumber(String contactNumber) {  this.contactNumber = contactNumber;  }  public double getCostPerDay() {  return costPerDay;  }  public void setCostPerDay(double costPerDay) {  this.costPerDay = costPerDay;  }  public String getOwnerName() {  return ownerName;  }  public void setOwnerName(String ownerName) {  this.ownerName = ownerName;  }  @Override  public String toString() {  return "Hall{" +  "name='" + name + '\'' +  ", contactNumber='" + contactNumber + '\'' +  ", costPerDay=" + costPerDay +  ", ownerName='" + ownerName + '\'' +  '}';  }  }  class Event6 {  private String name;  private String detail;  private String ownerName;  private String type;  public Event6(String name, String detail, String ownerName, String type) {  this.name = name;  this.detail = detail;  this.ownerName = ownerName;  this.type = type;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public String getDetail() {  return detail;  }  public void setDetail(String detail) {  this.detail = detail;  }  public String getOwnerName() {  return ownerName;  }  public void setOwnerName(String ownerName) {  this.ownerName = ownerName;  }  public String getType() {  return type;  }  public void setType(String type) {  this.type = type;  }  @Override  public String toString() {  return "Event6{" +  "name='" + name + '\'' +  ", detail='" + detail + '\'' +  ", ownerName='" + ownerName + '\'' +  ", type='" + type + '\'' +  '}';  }  }  class HallBooking {  private Hall hall;  private Event6 event;  private Date eventDate;  private double price;  public HallBooking(Hall hall, Event6 event, Date eventDate, double price) {  this.hall = hall;  this.event = event;  this.eventDate = eventDate;  this.price = price;  }  public Hall getHall() {  return hall;  }  public void setHall(Hall hall) {  this.hall = hall;  }  public Event6 getEvent() {  return event;  }  public void setEvent(Event6 event) {  this.event = event;  }  public Date getEventDate() {  return eventDate;  }  public void setEventDate(Date eventDate) {  this.eventDate = eventDate;  }  public double getPrice() {  return price;  }  public void setPrice(double price) {  this.price = price;  }  @Override  public String toString() {  return "HallBooking{" +  "hall=" + hall +  ", event=" + event +  ", eventDate=" + eventDate +  ", price=" + price +  '}';  }  }  class HallBookingBO {  public static boolean validateHallBooking(ArrayList<HallBooking> bookingList, HallBooking booking) throws HallNotAvailableException {  Set<Hall> hList = new HashSet<>();  Set<Event6> eList = new HashSet<>();  for (HallBooking hb : bookingList) {  hList.add(hb.getHall());  eList.add(hb.getEvent());  }  if (!hList.contains(booking.getHall()) || !eList.contains(booking.getEvent())){  throw new HallNotAvailableException("Event or Hall doesn't exist.");  }  for (HallBooking hb : bookingList) {  if (booking.getHall() == hb.getHall() && booking.getEventDate() == hb.getEventDate() && booking.getEvent() != hb.getEvent()) {  throw new HallNotAvailableException();  }  }  return true;  }  }  class HallNotAvailableException extends Exception {  public HallNotAvailableException() {  super("Hall Already Booked");  }  public HallNotAvailableException(String message) {  super(message);  }  } |
| --- |
|  |

Hardcode few Events and Halls in the driver class **Main**, get the bookings from the user and validate them, handle any exception thrown and finally display all bookings made.

**Note**: Hardcoded values are given in the template. Show message as "**Hall Already Booked**" for the already booked hall on the same date and "**Hall or Event does not exist**" for hall or event names not available in the list, in Custom exception.

Use **"%-15s%-15s%-15s%-15s"** to display the booking details in table format.

**Input and Output format:**

Refer to sample input and output for other further details and format of the output.

**[All Texts in bold corresponds to the input and rest are output] Sample Input and Output 1:**

Enter the booking details:

**Sdf hall,Book Fair,02-03-2018,15000**

Do you want to continue?(y/n)

**y**

Enter the booking details:

**Sdf hall,Furniture Fair,03-03-2018,16500**

Do you want to continue?(y/n)

**n**

The bookings entered are:

Hall id Event id Event date Price

Sdf hall Book Fair 02-03-2018 15000.0

Sdf hall Furniture Fair 03-03-2018 16500.0

**Sample Input and Output 2:**

Enter the booking details:

**TUV hall,Book Fair,01-02-1018,10000**

Do you want to continue?(y/n)

**y**

Enter the booking details:

**TUV hall,Furniture Fair,01-02-1018,15000**

HallNotAvailableException: Hall Already Booked

Do you want to continue?(y/n)

**y**

Enter the booking details:

**ABC hall,Furniture Fair,01-02-1018,10000**

HallNotAvailableException: Hall or Event does not exist

Do you want to continue?(y/n)

**y**

Enter the booking details:

**Sdf hall,Furniture Fair,01-02-1018,10000**

Do you want to continue?(y/n)

**n**

The bookings entered are:

Hall id Event id Event date Price

TUV hall Book Fair 01-02-1018 10000.0

Sdf hall Furniture Fair 01-02-1018 10000.0

**Custom Exception - Bank Withdrawal**

A customer goes to the ATM to withdraw money from his account. The maximum withdrawal limit in ICICI bank for a day is Rs. 25,000.

Write a program that accepts the withdrawal amount from the customer as input. When the amount entered by the user is greater than the withdrawal limit (Rs. 25000), a custom exception named **MinimumAccountBalance** is thrown. Use exception handling mechanisms to handle this exception.

**Input Format:**

The input consists of a double value which corresponds to the amount to withdraw.

**Output Format:**

If the amount to withdraw is less than Rs.25,000 print “**Amount Withdrawn Successfully**”. Else throw the custom exception.

Refer to sample input and output for formatting specifications.

**Note: All text in bold corresponds to input and the rest corresponds to output.**

**Sample Input and Output 1:**

Enter amount to withdrawal

**5000**

Amount Withdrawn Successfully

**Sample Input and Output 2:**

Enter amount to withdrawal

**26000**

Caught MinimumAccountBalance: Minimum Account Balance Exception

| import java.lang.\*;  import java.util.\*;  public class Main {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.println("Enter amount for withdrawal: ");  double amount = sc.nextDouble();  try {  if (validate(amount)) {  System.out.println("Amount withdrawn successfully.");  }  } catch (MinimumAccountBalance e) {  System.err.println("Caught " + e);  }  }  public static boolean validate (double amt) throws MinimumAccountBalance{  if (amt > -1 && amt < 25001) {  return true;  }  else {  throw new MinimumAccountBalance("Minimum Account Balance Exception");  }  }  }  class MinimumAccountBalance extends Exception {  public MinimumAccountBalance(String message) {  super(message);  }  } |
| --- |
|  |