Info

- **Name** : Yash Kumar Verma
- **Registration Number**: 19BCE2669
- **Link to source code**: YashKumarVerma/java-multithreading-exceptions
- **Problem Statement** :

41	BCE	19BCE2669	YASH KUMAR VERMA	user defined exceptions - Multithreading

- Question Division
  - **MultiThreading** : Question1 to Question6
  - **Exceptions** : Question7 to Question10

A note about submission

- All programs use a package **rainbow** written by me to show colored and formatted output. The source of the rainbow package can be found in `rainbow/rainbow.java` file in any question directory.
- In sample outputs, you can often see a green text enclosed towards the left hand side, this shows the thread name in multi-threading programs and class name / block name in programs demonstrating user defined exceptions.

```
[ main ] Enter url of the page you want to save : https://github.com/yashkumarverma/
[ main ] Enter name of file you want to save webpage as :home.html
[ Thread-0 ] File Downloader Thread initialized
[ main ] Your file is downloading now.
[ Thread-1 ] File Writer Thread initialized
[ Thread-1 ] Data written to file : home.html
```

Question1

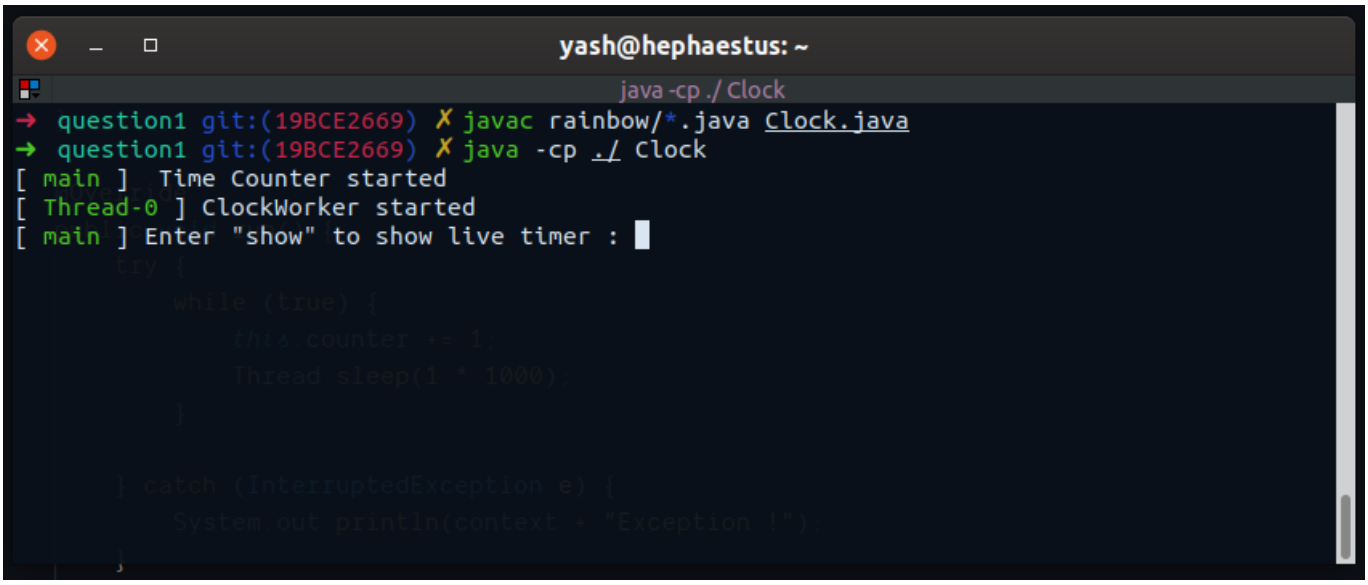
Implementing a time counter via a separate thread, and call the function whenever user asks to show the time.

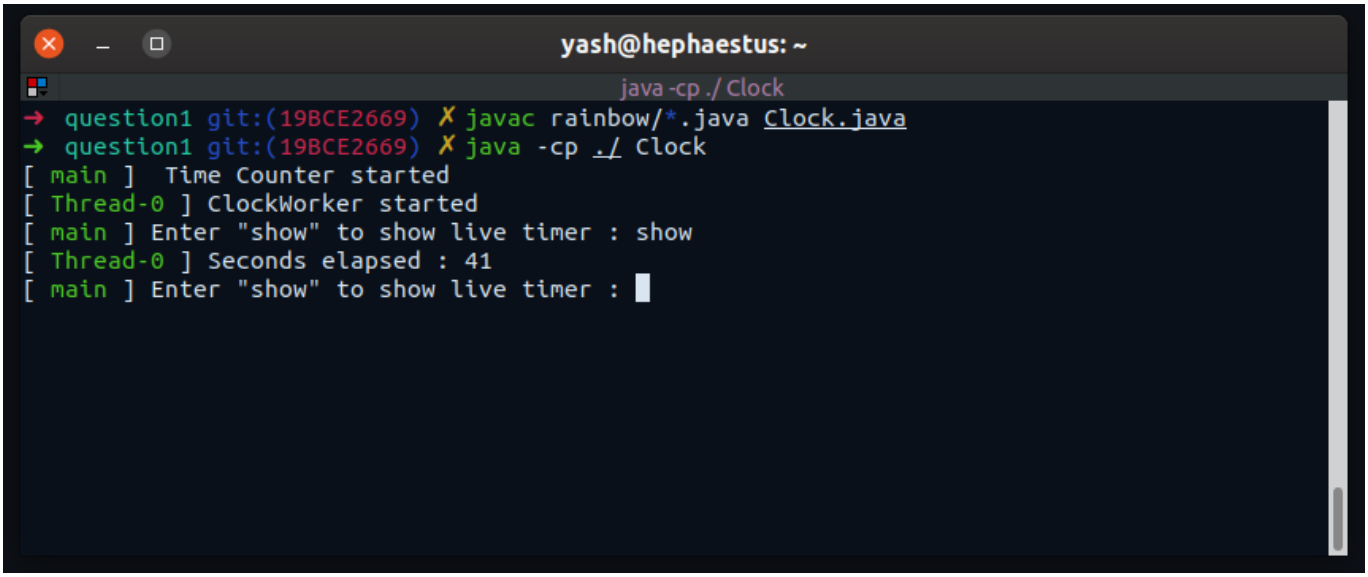
Code

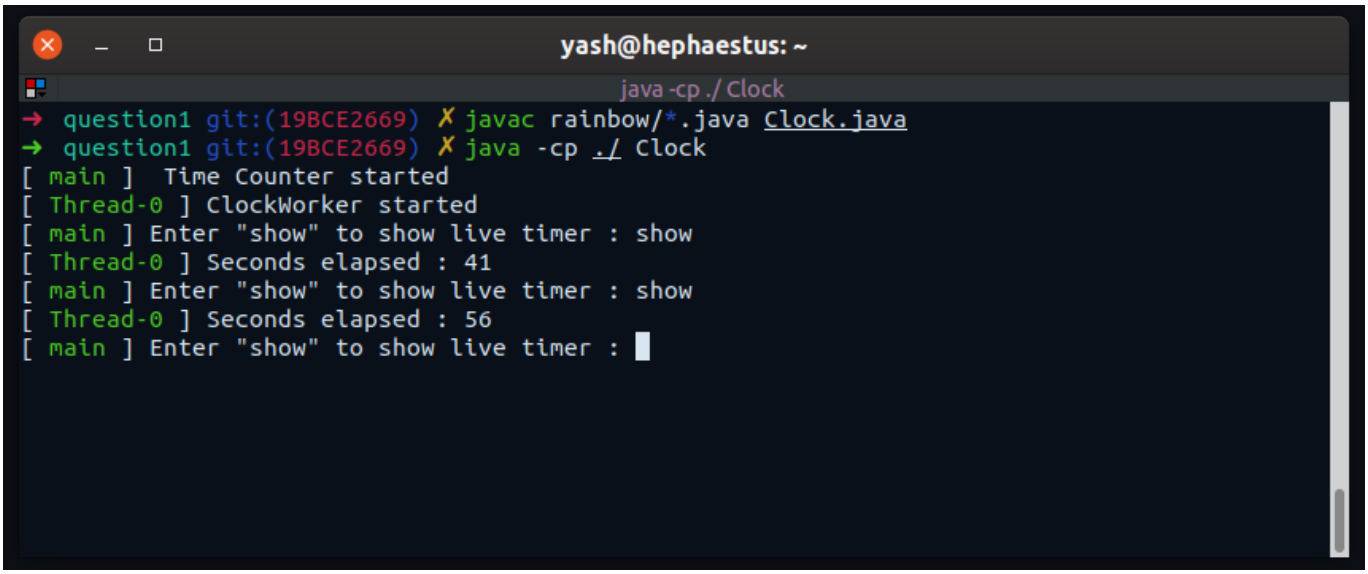
```
import java.util.Scanner;
import rainbow.rainbow;

class ClockWorker extends Thread {
    private int counter = 0;
    private final String context = "[ " + rainbow.green(this.getName()) + "
] ";

    ClockWorker() {
        System.out.println(this.context + "ClockWorker started");
    }

    @Override
```

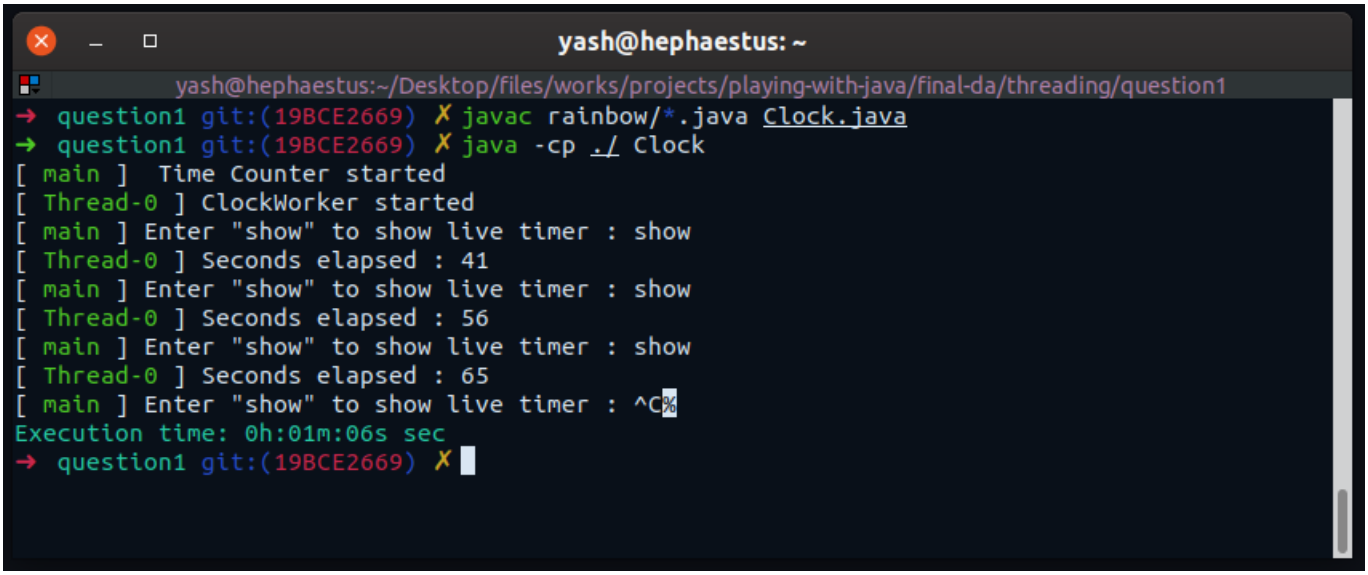
```
    public void run() {
        try {
            while (true) {
                this.counter += 1;
                Thread.sleep(1 * 1000);
            }

        } catch (InterruptedException e) {
            System.out.println(context + "Exception !");
        }
    }

    public int getSeconds() {
        System.out.print(context);
        return this.counter;
    }
}

public class Clock {

    public static void main(String args[]) {
        String command;
        final Scanner handler = new Scanner(System.in);
        final String context = "[ " +
rainbow.green(Thread.currentThread().getName()) + " ] ";

        // tell user about current thread and that counter has started
        System.out.println(context + " Time Counter started");

        // start separate thread with timer
        final ClockWorker worker = new ClockWorker();
        worker.start();

        while (true) {
            System.out.print(context + "Enter \"show\" to show live timer :
");
            command = handler.nextLine();
            if (command.equals("show")) {
                System.out.println("Seconds elapsed : " +
worker.getSeconds());
            }
        }

    }
}
```

Output


yash@hephaestus: ~
java -cp ./ Clock
➜ question1 git:(19BCE2669) ✗ javac rainbow/*.java Clock.java
➜ question1 git:(19BCE2669) ✗ java -cp ./ Clock
[main] Time Counter started
[Thread-0] ClockWorker started
[main] Enter "show" to show live timer :


yash@hephaestus: ~
java -cp ./ Clock
➜ question1 git:(19BCE2669) ✗ javac rainbow/*.java Clock.java
➜ question1 git:(19BCE2669) ✗ java -cp ./ Clock
[main] Time Counter started
[Thread-0] ClockWorker started
[main] Enter "show" to show live timer : show
[Thread-0] Seconds elapsed : 41
[main] Enter "show" to show live timer :





Question2

Implementing two threads to calculate the nth prime number and prime factors of a number and also show the current threads being executed.

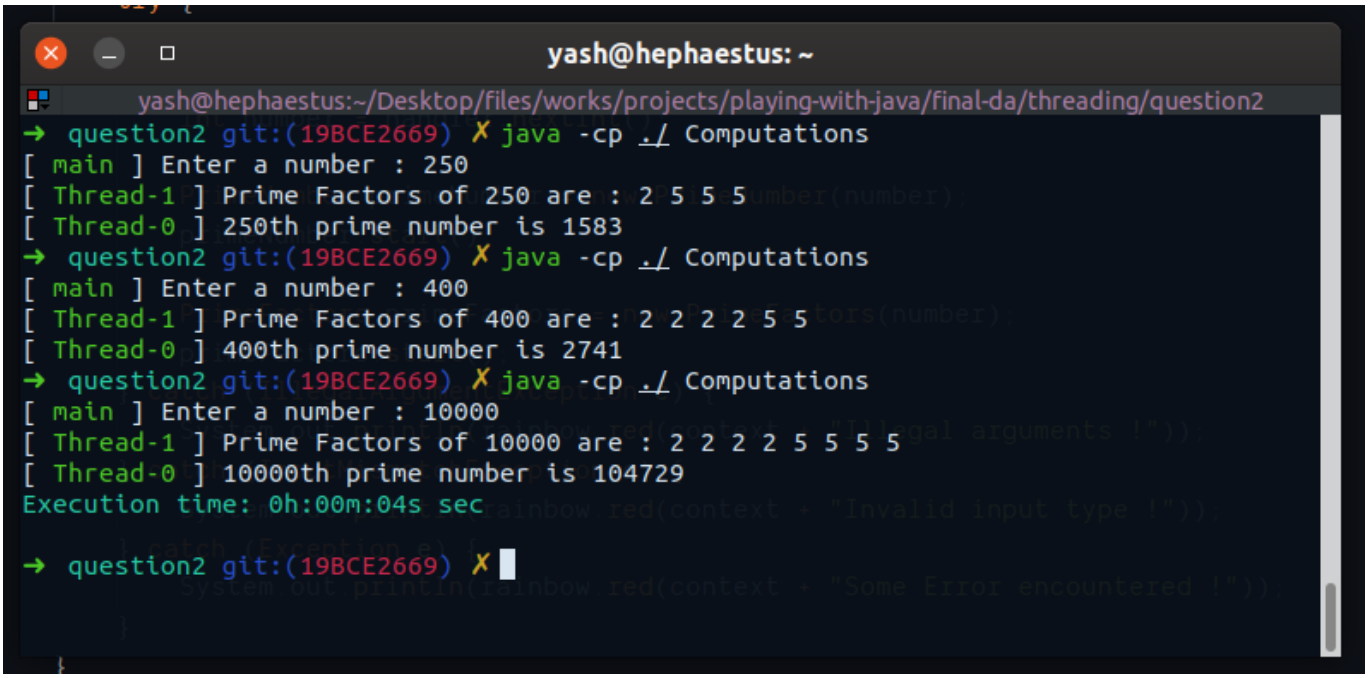
Code

```
import java.util.Scanner;
import java.util.ArrayList;
import java.util.InputMismatchException;
import java.util.List;
import rainbow.rainbow;
import rainbow.rainbow;

class PrimeNumber extends Thread {
    private int nth;
    private final String context = "[ " + rainbow.green(this.getName()) + "
] ";
```

```
    public PrimeNumber(int nth) {
        this.nth = nth;
    }

    @Override
    public void run() {
        System.out.println(this.context + this.nth + "th prime number is "
+ this.findNth(this.nth));
    }

    private int findNth(int nth) {
        int num, count, i;
        num = 1;
        count = 0;

        while (count < nth) {
            num = num + 1;
            for (i = 2; i <= num; i++) {
                if (num % i == 0) {
                    break;
                }
            }
            if (i == num) {
                count = count + 1;
            }
        }

        return num;
    }
}

class PrimeFactors extends Thread {
    private int number;
    private Integer[] factors;
    private final String context = "[ " + rainbow.green(this.getName()) + "
] ";

    public PrimeFactors(int number) {
        this.number = number;
    }

    @Override
    public void run() {
        System.out.print(this.context + "Prime Factors of " + this.number +
" are : ");
        this.calculator();
        for (Integer num : this.factors) {
            System.out.print(num + " ");
        }
        System.out.println();
    }

    private void calculator() {
        List<Integer> list = new ArrayList<Integer>();
```

```
        for (int i = 2; i < this.number; i++) {
            while (this.number % i == 0) {
                list.add(i);
                this.number = this.number / i;
            }
        }
        this.factors = list.toArray(new Integer[list.size()]);
    }
}

public class Computations {
    public static void main(String args[]) {
        final String context = "[ " +
rainbow.green(Thread.currentThread().getName()) + " ] ";

        try {
            Scanner handler = new Scanner(System.in);
            System.out.print(context + rainbow.bold("Enter a number : "));
            int number = handler.nextInt();

            PrimeNumber primeNumber = new PrimeNumber(number);
            primeNumber.start();

            PrimeFactors primeFactors = new PrimeFactors(number);
            primeFactors.start();
        } catch (IllegalArgumentException e) {
            System.out.println(rainbow.red(context + "Illegal arguments
!"));
        } catch (InputMismatchException e) {
            System.out.println(rainbow.red(context + "Invalid input type
!"));
        } catch (Exception e) {
            System.out.println(rainbow.red(context + "Some Error
encountered !"));
        }
    }
}
```

Output



Question3

Create two threads. One to make a web request and download the html of a page, and another to write the data to file.

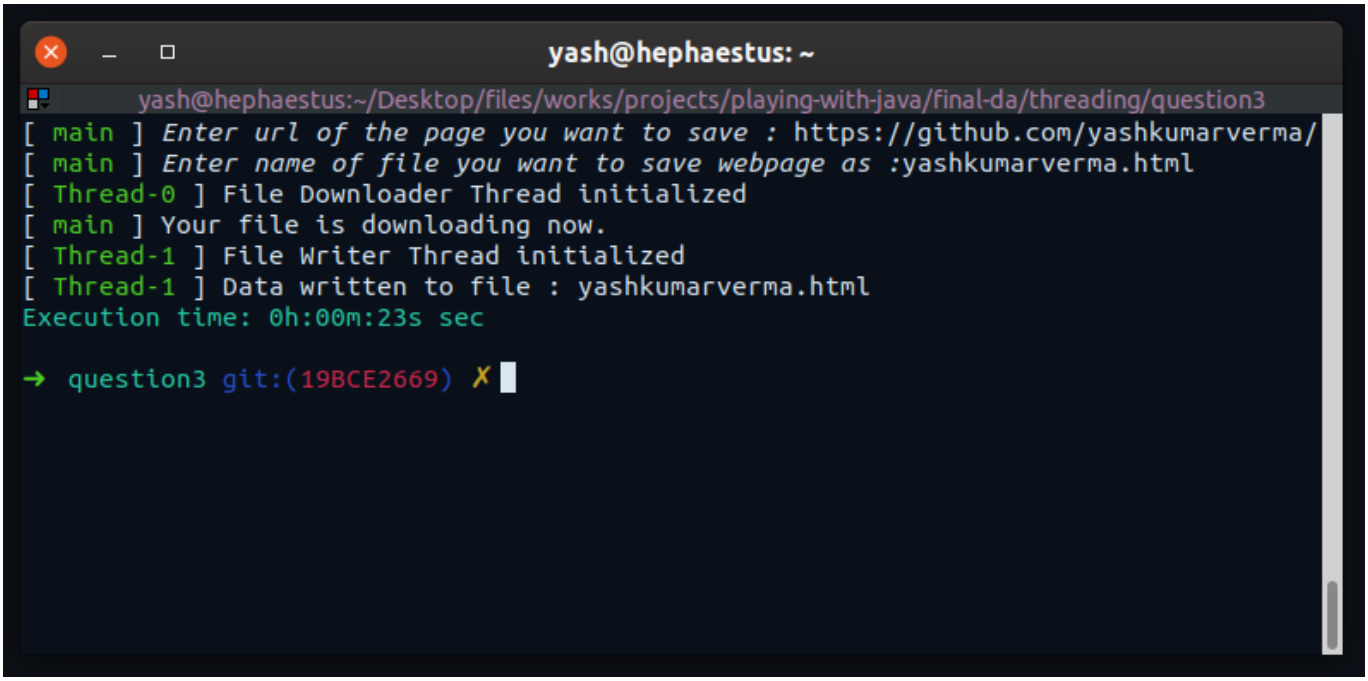
Code

```
import java.net.URI;
import java.util.Scanner;
import java.io.FileWriter;
import java.io.IOException;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import rainbow.rainbow;

class FileWriterThread extends Thread {
    private String data;
    private String fileName;
    private FileWriter fileWriter = null;
    private final String context = "[ " + rainbow.green(this.getName()) + "
] ";

    public FileWriterThread(String fileName, String data) throws
IOException {
        System.out.println(this.context + "File Writer Thread
initialized");
        this.data = data;
        this.fileName = fileName;
        this.fileWriter = new FileWriter("./" + fileName);
    }
```

```
    @Override
    public void run() {
        try {
            this.fileWriter.write(this.data);
            System.out.println(this.context + "Data written to file : " +
this.fileName);
        } catch (IOException e) {
            System.out.print(e);
        }
    }

    @Override
    protected void finalize() throws Throwable {
        System.out.println(this.context + "File Writer closed.");
        this.fileWriter.close();
    }
}

class FileDownloader extends Thread {
    private String fileUrl;
    private String fileName;
    private final String context = "[ " + rainbow.green(this.getName()) + "
] ";

    public FileDownloader(String fileUrl, String fileName) {
        System.out.println(this.context + "File Downloader Thread
initialized");
        this.fileUrl = fileUrl;
        this.fileName = fileName;
    }

    public void run() {
        try {
            final HttpClient client = HttpClient.newHttpClient();
            final HttpRequest request =
HttpRequest.newBuilder().uri(URI.create(this.fileUrl)).build();
            final HttpResponse<String> response = client.send(request,
HttpResponse.BodyHandlers.ofString());
            final String dataToWriteToFile = response.body();
            final FileWriterThread fileWriterThread = new
FileWriterThread(this.fileName, dataToWriteToFile);
            fileWriterThread.start();

        } catch (Exception e) {
            System.out.println(this.context + "Error downloading webpage");
        }
    }
}

public class FileWorker {

    public static void main(final String[] args) {
        final Scanner handler = new Scanner(System.in);
        final String context = "[ " +
```

```
  rainbow.green(Thread.currentThread().getName()) + " ] ";

          // take input of url
          System.out.print(context + rainbow.italic("Enter url of the page
  you want to save : "));
          String url = handler.nextLine();

          // take input of filename
          System.out.print(context + rainbow.italic("Enter name of file you
  want to save webpage as :"));
          String fileName = handler.nextLine();

          // download file
          FileDownloader fileDownloader = new FileDownloader(url, fileName);
          fileDownloader.start();

          System.out.println(context + "Your file is downloading now.");

          handler.close();
      }
  }
```

Output



Content of file : "**yashkumarverma.html**"

Question4

Write a JAVA Program to create parallel connections to database using JDBC driver, and insert data into database via different threads.

Code

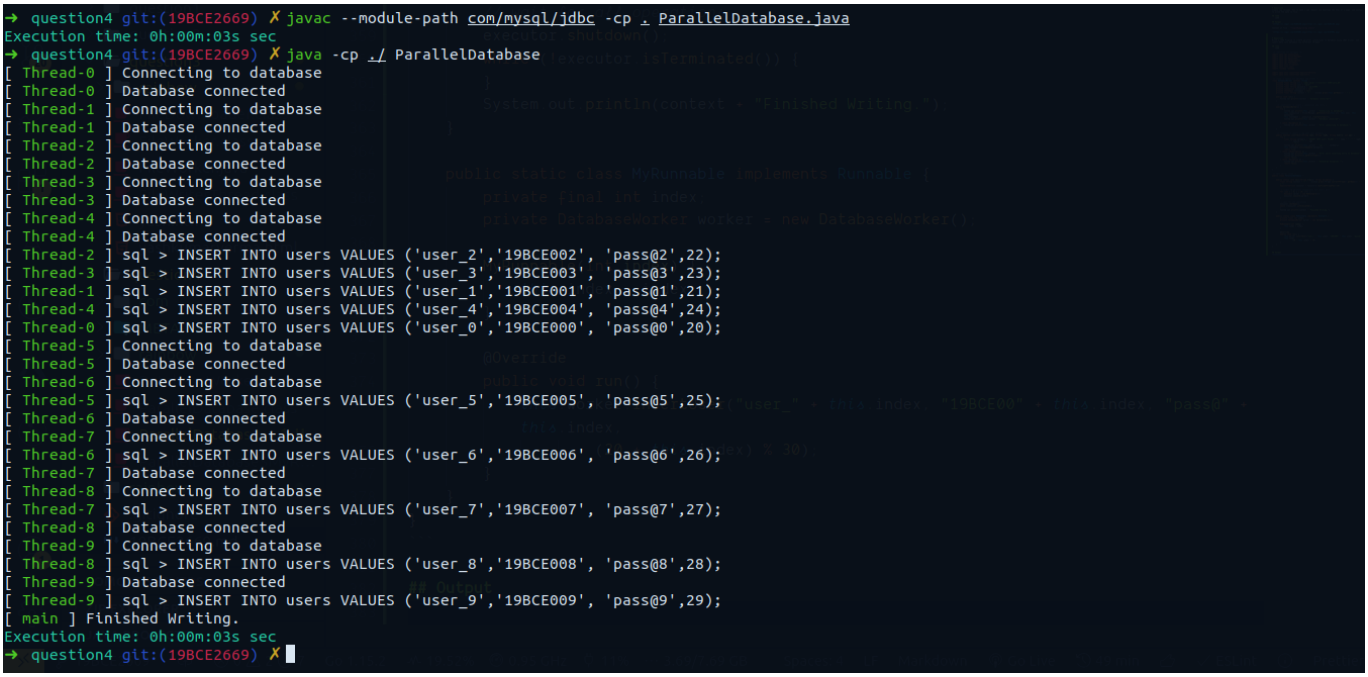
```
/** libraries for database **/
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import rainbow.rainbow;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

/** class to handle all database operations */
class DatabaseWorker extends Thread {
    private final String url = "jdbc:mysql://localhost:3306/java_db";
    private final String user = "yash";
    private final String password = "yash2000.";
    private Connection connection = null;
    private Statement statement = null;
    private final String context = "[ " + rainbow.green(this.getName()) + "
] ";
```

```
    public void run() {
        System.out.println(context + "database connected");
    }

    /** constructor to create a connection and store */
    public DatabaseWorker() {
        try {
            System.out.println(this.context + "Connecting to database");
            this.connection = DriverManager.getConnection(this.url,
this.user, this.password);
            this.statement = connection.createStatement();
            System.out.println(this.context + "Database connected");

        } catch (Exception e) {
            System.out.println(this.context + "Error connecting to
database;");
        }
    }

    /** function to insert a new user into database */
    public boolean insertUser(String name, String regNo, String mobile, int
age) {
        try {
            final String sqlQuery = "INSERT INTO users VALUES ('" + name +
"','" + regNo + "', '" + mobile + "'," + age
                    + ");";
            System.out.println(this.context + "sql > " + sqlQuery);
            this.statement.executeUpdate(sqlQuery);
            return true;
        } catch (SQLException e) {
            System.out.println(this.context + "Error while inserting entry
to database");
            System.out.println(e.getErrorCode());
            return false;
        } catch (Exception e) {
            System.out.println(this.context + "Unhandled Exception !");
            return false;
        }
    }
}

/** main worker class */
public class ParallelDatabase {

    public static void main(String args[]) throws Exception {
        final String context = "[ " +
rainbow.green(Thread.currentThread().getName()) + " ] ";
        ExecutorService executor = Executors.newFixedThreadPool(10);

        // create 10 parallel connections and insert data
        for (int i = 0; i < 10; i++) {
            Runnable worker = new MyRunnable(i);
            executor.execute(worker);
        }
```

```
            // wait for all operations
            executor.shutdown();
            while (!executor.isTerminated()) {
            }
            System.out.println(context + "Finished Writing.");
        }

        public static class MyRunnable implements Runnable {
            private final int index;
            private DatabaseWorker worker = new DatabaseWorker();

            MyRunnable(int index) {
                this.index = index;
            }

            @Override
            public void run() {
                this.worker.insertUser("user_" + this.index, "19BCE00" +
    this.index, "pass@" + this.index,
                        (20 + this.index) % 30);
            }
        }
    }
```

Output

Commands executed by each thread have the thread name towards the left hand side in the format of **Thread-*{threadNumber}***



Question5

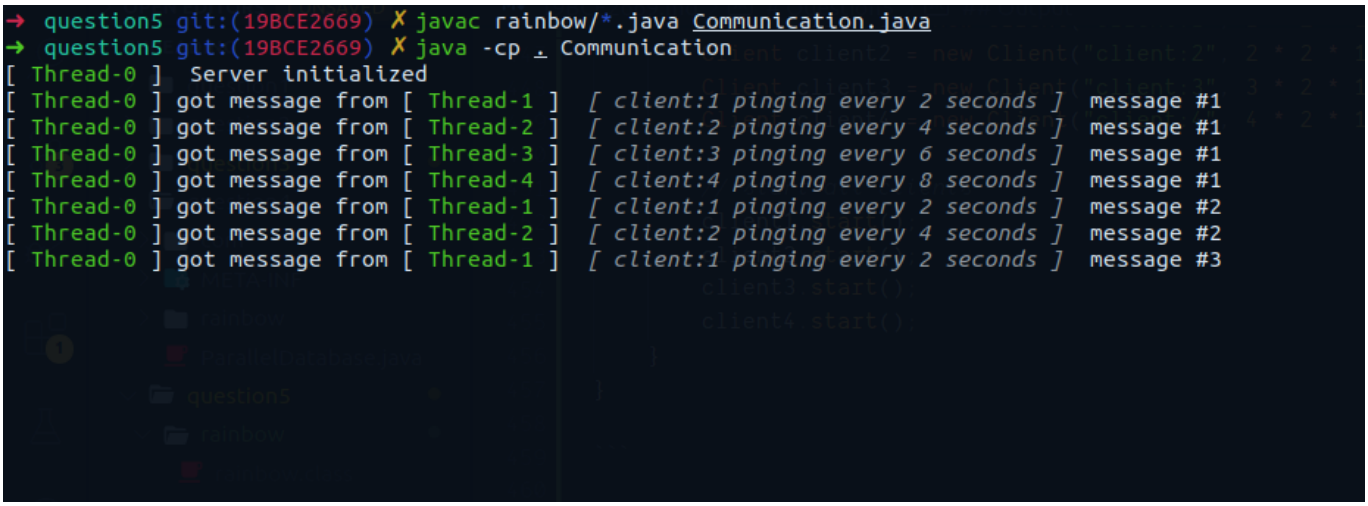
Write a JAVA program to simulate a client server architecture with server occupying a thread and each client being a different thread. Send messages from clients to server at regular intervals of time.

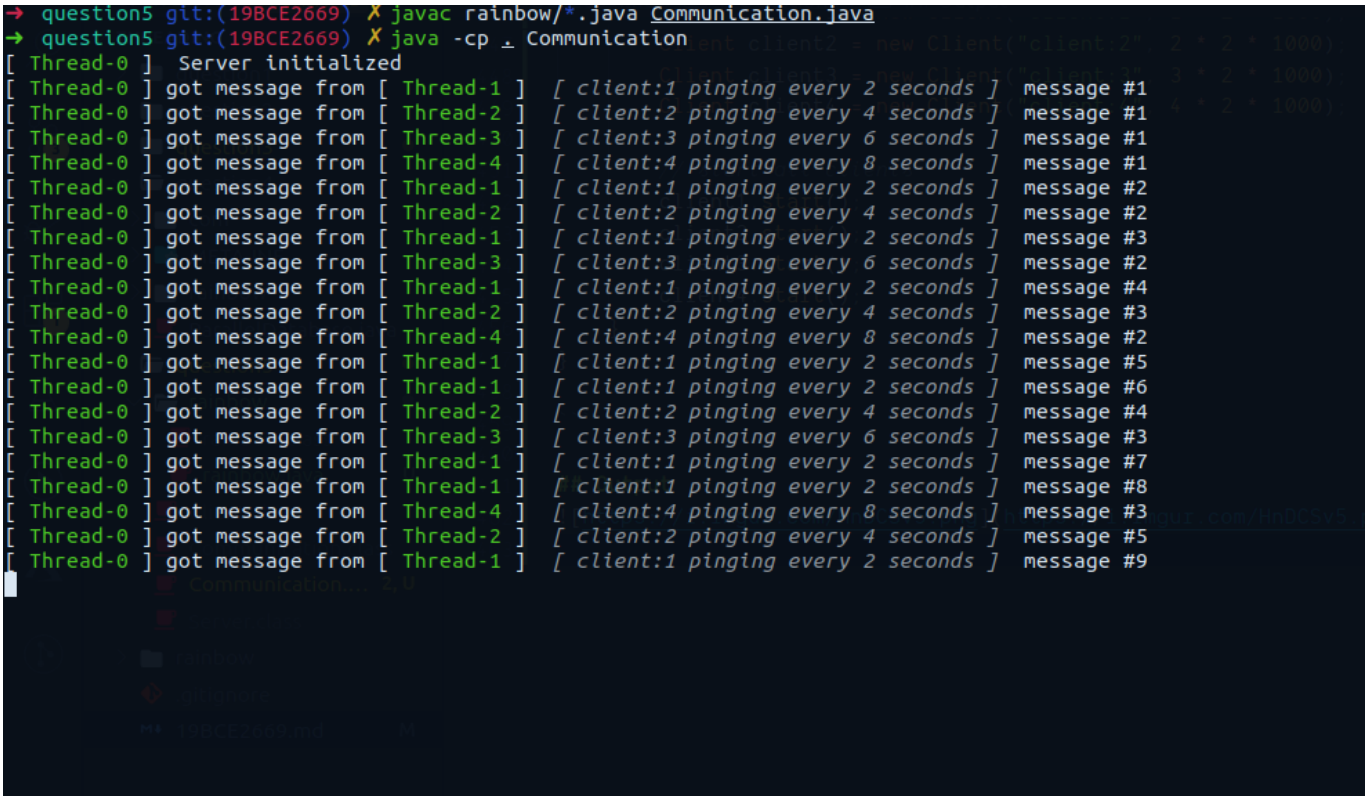
Code

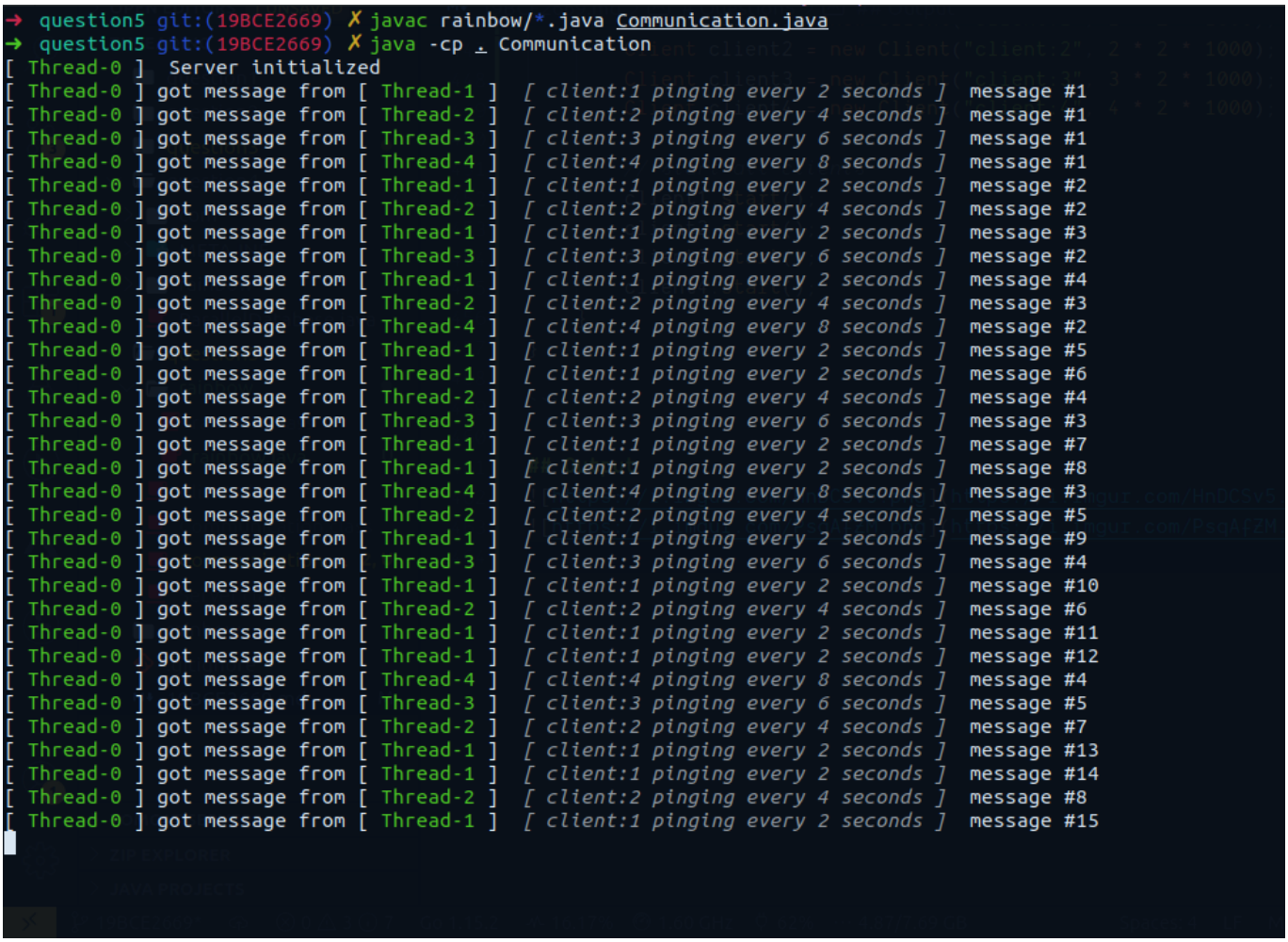
```
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
import rainbow.rainbow;

class Server extends Thread {
    private final String context = "[ " + rainbow.green(this.getName()) + "
] ";
    public static BlockingQueue<String> messages = new
LinkedBlockingQueue<String>();

    public Server() {
        System.out.println(this.context + " Server initialized");
    }

    public void run() {
        while (true) {
            String message;
            while ((message = Server.messages.poll()) != null) {
                System.out.println(this.context + "got message from " +
message);
            }
        }
    }
}

class Client extends Thread {
    private String context;
    private String name;
    private int coolDownTime;

    public Client(String name, int coolDownTime) {
        this.name = name;
        this.coolDownTime = coolDownTime;
        this.context = "[ " + rainbow.green(this.getName()) + " ] " +
rainbow
                .dim(rainbow.italic(" [ " + this.name + " pinging every " +
this.coolDownTime / 1000 + " seconds ] "));
    }

    public void run() {
        int messageNumber = 1;
        while (true) {
            try {
                Server.messages.put(this.context + " message #" +
messageNumber);
                messageNumber++;
                this.sleep(this.coolDownTime);
```

```
                } catch (InterruptedException e) {
                    System.out.println(this.context + "Error Sending message");
                }
            }
        }
    }

    public class Communication {
        public static void main(String args[]) {
            Server server = new Server();
            server.start();

            Client client1 = new Client("client:1", 1 * 2 * 1000);
            Client client2 = new Client("client:2", 2 * 2 * 1000);
            Client client3 = new Client("client:3", 3 * 2 * 1000);
            Client client4 = new Client("client:4", 4 * 2 * 1000);

            // start all clients
            client1.start();
            client2.start();
            client3.start();
            client4.start();
        }
    }
```

Output


→ question5 git:(19BCE2669) ✗ javac rainbow/*.java Communication.java
→ question5 git:(19BCE2669) ✗ java -cp . Communication
[Thread-0] Server initialized
[Thread-0] got message from [Thread-1] [client:1 pinging every 2 seconds] message #1
[Thread-0] got message from [Thread-2] [client:2 pinging every 4 seconds] message #1
[Thread-0] got message from [Thread-3] [client:3 pinging every 6 seconds] message #1
[Thread-0] got message from [Thread-4] [client:4 pinging every 8 seconds] message #1
[Thread-0] got message from [Thread-1] [client:1 pinging every 2 seconds] message #2
[Thread-0] got message from [Thread-2] [client:2 pinging every 4 seconds] message #2
[Thread-0] got message from [Thread-1] [client:1 pinging every 2 seconds] message #3


→ question5 git:(19BCE2669) ✗ javac rainbow/*.java Communication.java
→ question5 git:(19BCE2669) ✗ java -cp . Communication
[Thread-0] Server initialized
[Thread-0] got message from [Thread-1] [client:1 pinging every 2 seconds] message #1
[Thread-0] got message from [Thread-2] [client:2 pinging every 4 seconds] message #1
[Thread-0] got message from [Thread-3] [client:3 pinging every 6 seconds] message #1
[Thread-0] got message from [Thread-4] [client:4 pinging every 8 seconds] message #1
[Thread-0] got message from [Thread-1] [client:1 pinging every 2 seconds] message #2
[Thread-0] got message from [Thread-2] [client:2 pinging every 4 seconds] message #2
[Thread-0] got message from [Thread-1] [client:1 pinging every 2 seconds] message #3
[Thread-0] got message from [Thread-3] [client:3 pinging every 6 seconds] message #2
[Thread-0] got message from [Thread-1] [client:1 pinging every 2 seconds] message #4
[Thread-0] got message from [Thread-2] [client:2 pinging every 4 seconds] message #3
[Thread-0] got message from [Thread-4] [client:4 pinging every 8 seconds] message #2
[Thread-0] got message from [Thread-1] [client:1 pinging every 2 seconds] message #5
[Thread-0] got message from [Thread-1] [client:1 pinging every 2 seconds] message #6
[Thread-0] got message from [Thread-2] [client:2 pinging every 4 seconds] message #4
[Thread-0] got message from [Thread-3] [client:3 pinging every 6 seconds] message #3
[Thread-0] got message from [Thread-1] [client:1 pinging every 2 seconds] message #7
[Thread-0] got message from [Thread-1] [client:1 pinging every 2 seconds] message #8
[Thread-0] got message from [Thread-4] [client:4 pinging every 8 seconds] message #3
[Thread-0] got message from [Thread-2] [client:2 pinging every 4 seconds] message #5
[Thread-0] got message from [Thread-1] [client:1 pinging every 2 seconds] message #9



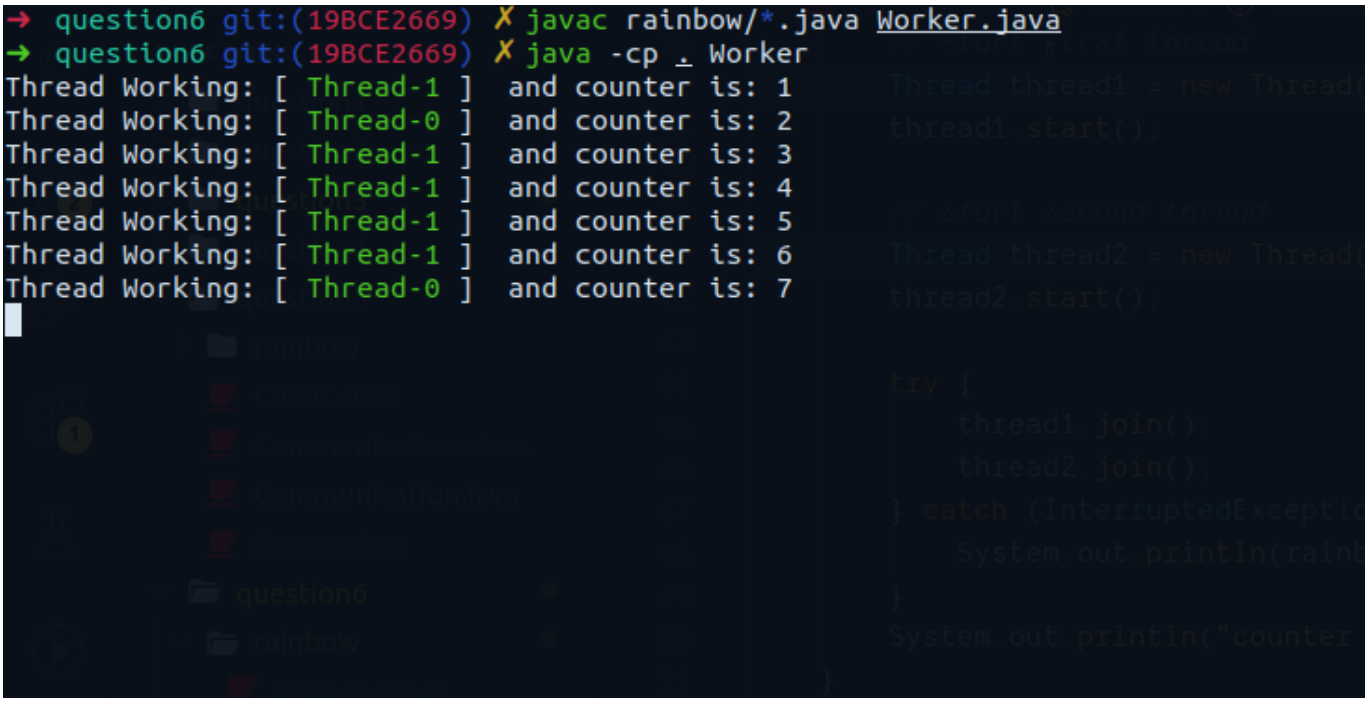
Question6

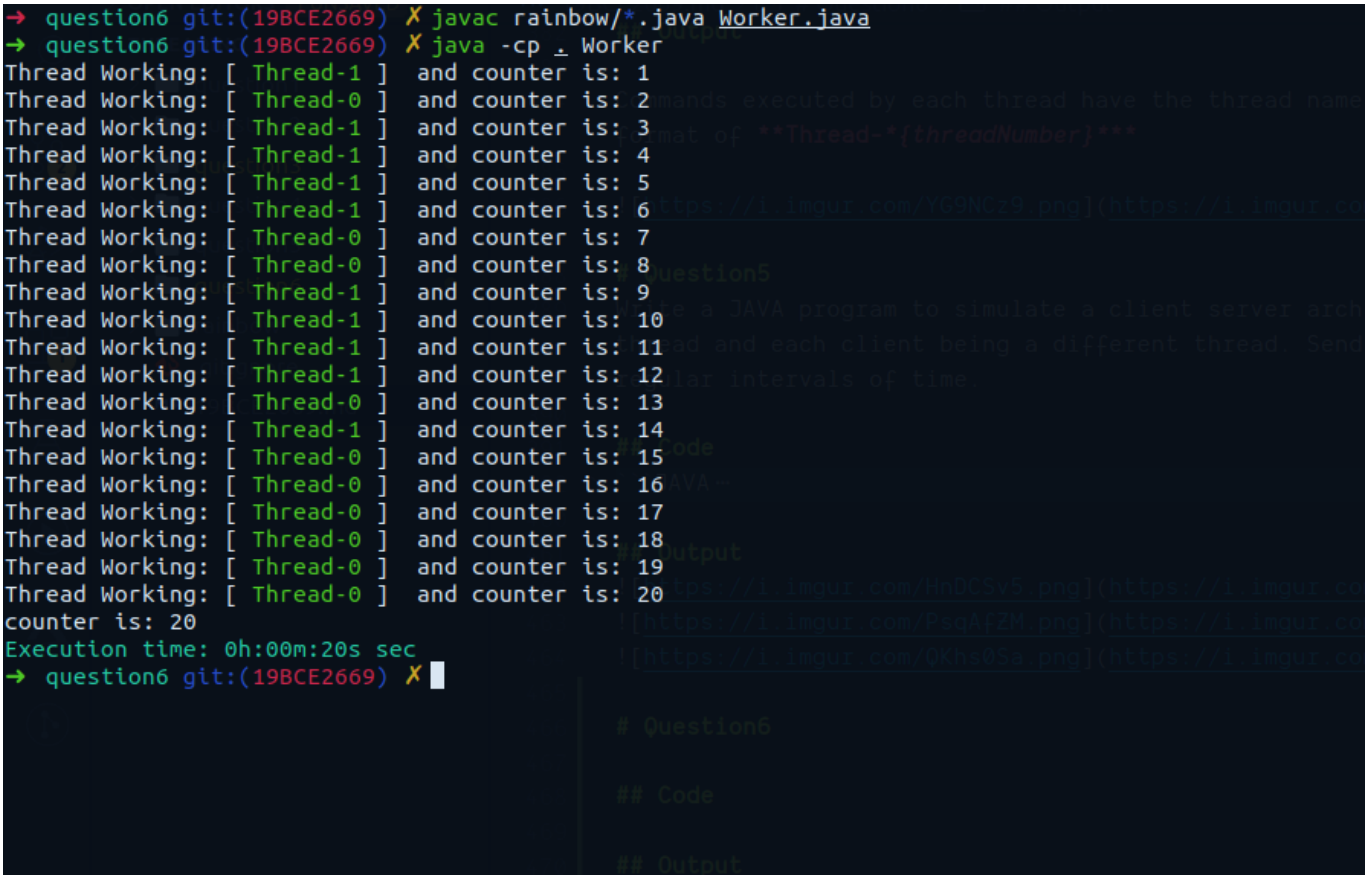
Code

```
import rainbow.rainbow;

public class Worker {
    private int counter = 0;

    // synchronized keyword to allow only one thread to execute at one time
    public synchronized void increment(String threadName) throws
InterruptedException {
        counter++;
        Thread.sleep(1000);
        System.out.println("Thread Working: " + threadName + " and counter
is: " + counter);
    }

    // function to invoke clients
    public void doWork() {

        // start first thread
        Thread thread1 = new Thread(new Runnable() {
            public void run() {
```

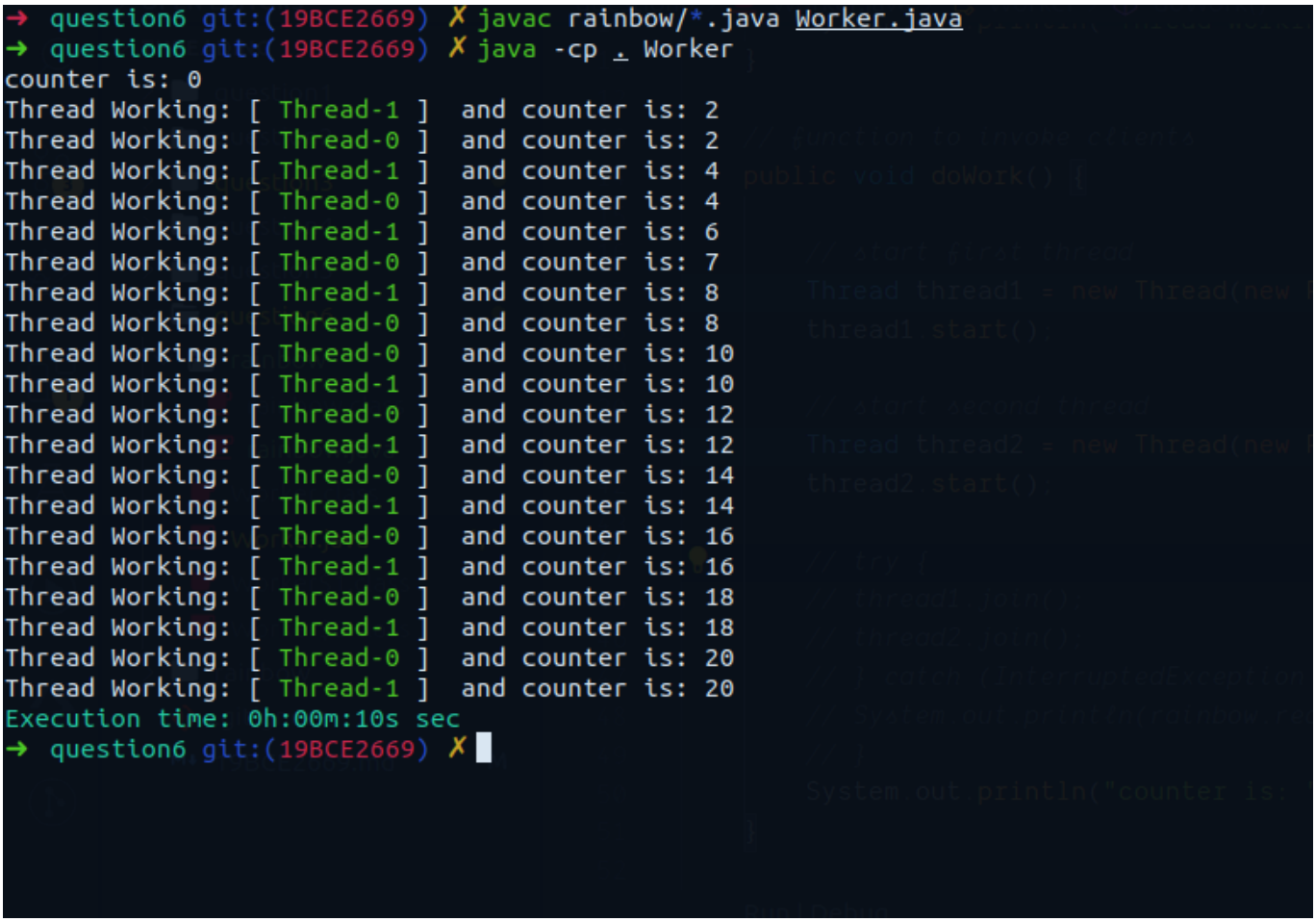
```
                for (int i = 0; i < 10; i++) {
                    try {
                        increment("[ " +
rainbow.green(Thread.currentThread().getName()) + " ] ");
                    } catch (InterruptedException ex) {
                        System.out.println("Error !");
                    }
                }
            }
        });
        thread1.start();

        // start second thread
        Thread thread2 = new Thread(new Runnable() {
            public void run() {
                for (int i = 0; i < 10; i++) {
                    try {
                        increment("[ " +
rainbow.green(Thread.currentThread().getName()) + " ] ");
                    } catch (InterruptedException ex) {
                        System.out.println("Error !");
                    }
                }
            }
        });
        thread2.start();

        try {
            thread1.join();
            thread2.join();
        } catch (InterruptedException ignored) {
            System.out.println(rainbow.red("Couldn't join threads"));
        }
        System.out.println("counter is: " + counter);
    }

    public static void main(String[] args) {
        Worker worker = new Worker();
        worker.doWork();
    }
}
```

Output





When we don't use synchronized keyword, the threads execute the function at the same time, and we see undesirable results.



Question7

Write a JAVA Program to show usage of custom - user defined exceptions and override the base methods to change the way error messages are shown to user.

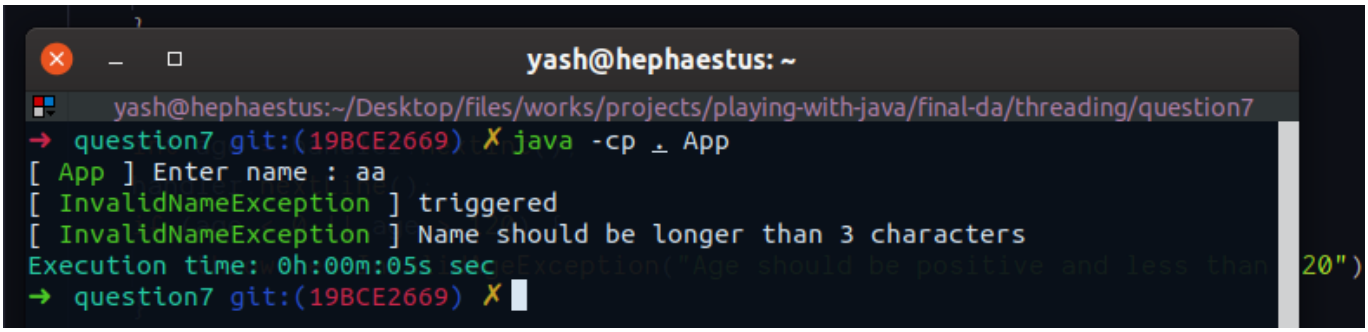
Code

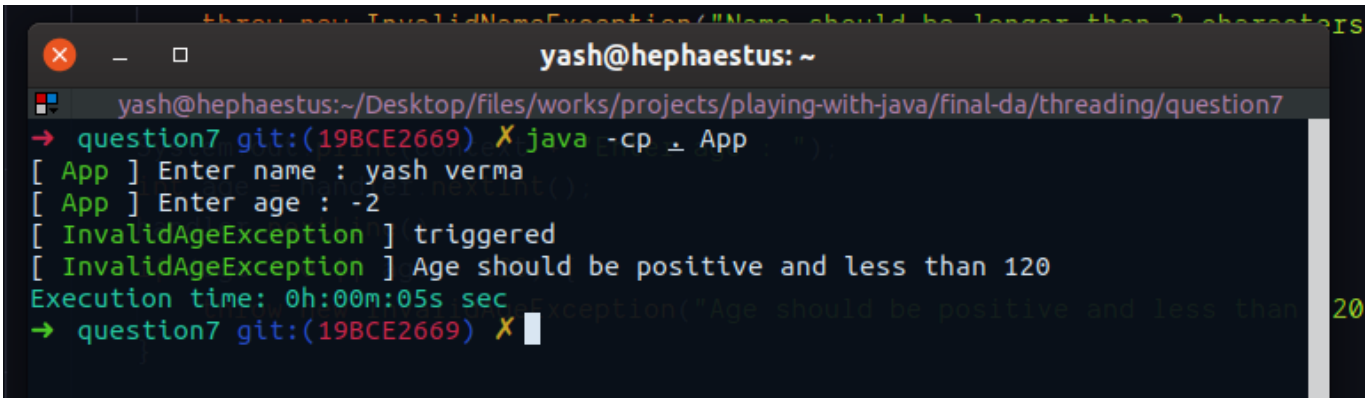
```
import java.util.InputMismatchException;
import java.util.Scanner;
import rainbow.rainbow;

class InvalidNameException extends Exception {
    private final String context = "[ " +
rainbow.green("InvalidNameException") + " ] ";
    private String message;

    public InvalidNameException(String message) {
        super(message);
        this.message = message;
        System.out.println(this.context + "triggered");
    }

    public String getMessage() {
        return this.context + this.message;
    }
```

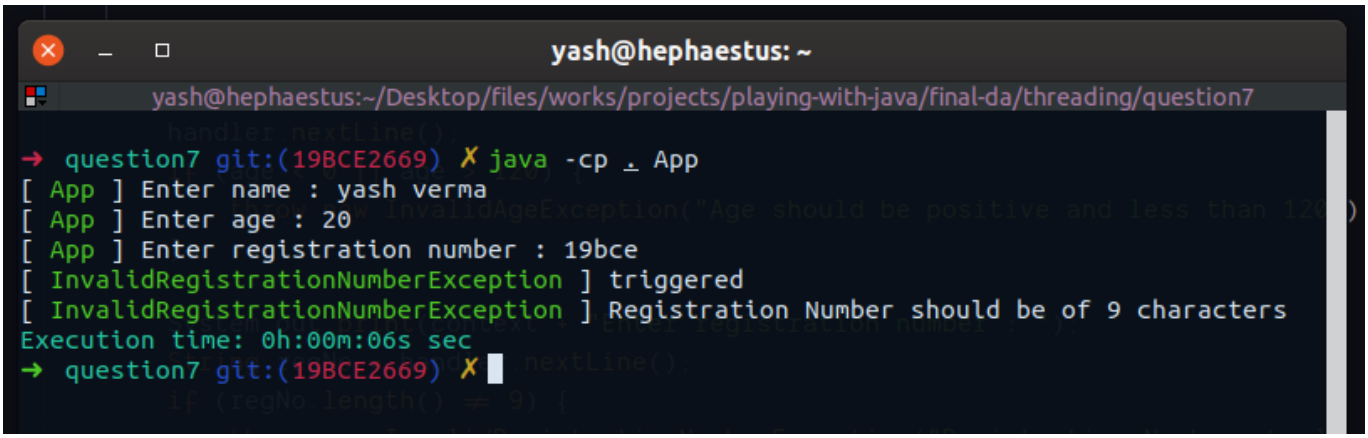
```
};

class InvalidAgeException extends Exception {
    private final String context = "[ " +
rainbow.green("InvalidAgeException") + " ] ";
    private String message;

    public InvalidAgeException(String message) {
        super(message);
        this.message = message;
        System.out.println(this.context + "triggered");
    }

    public String getMessage() {
        return this.context + this.message;
    }
};

class InvalidRegistrationNumberException extends Exception {
    private final String context = "[ " +
rainbow.green("InvalidRegistrationNumberException") + " ] ";
    private String message;

    public InvalidRegistrationNumberException(String message) {
        super(message);
        this.message = message;
        System.out.println(this.context + "triggered");
    }

    public String getMessage() {
        return this.context + this.message;
    }
};

public class App {
    public static void main(String args[]) {
        final String context = "[ " + rainbow.green("App") + " ] ";

        try {
            Scanner handler = new Scanner(System.in);
            System.out.print(context + "Enter name : ");
            String name = handler.nextLine();
            if (name.length() <= 3) {
                throw new InvalidNameException("Name should be longer than
3 characters");
            }

            System.out.print(context + "Enter age : ");
            int age = handler.nextInt();
            handler.nextLine();
            if (age < 0 || age > 120) {
                throw new InvalidAgeException("Age should be positive and
less than 120");
            }
```

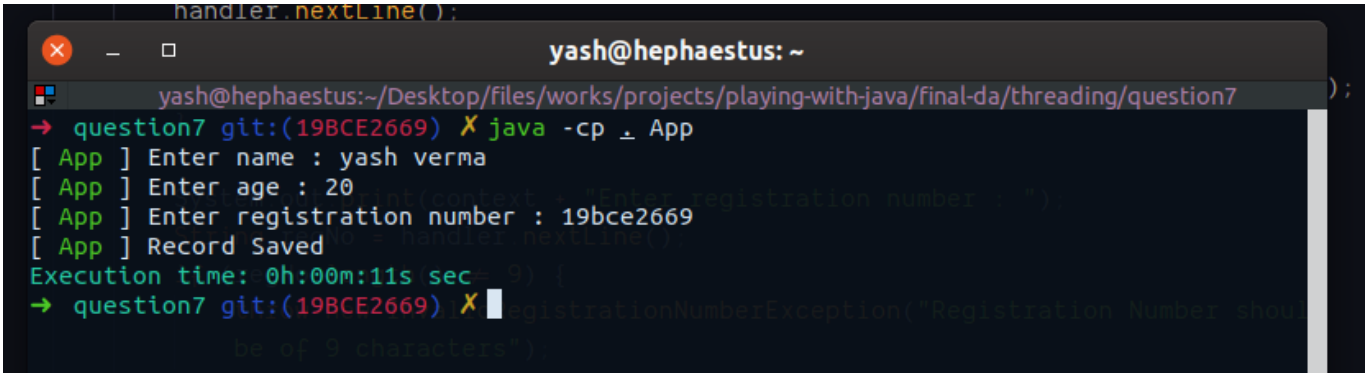
```
            System.out.print(context + "Enter registration number : ");
            String regNo = handler.nextLine();
            if (regNo.length() != 9) {
                throw new InvalidRegistrationNumberException("Registration
Number should be of 9 characters");
            }

            System.out.println(context + "Record Saved");

        } catch (InvalidNameException e) {
            System.out.println(e.getMessage());
        } catch (InvalidAgeException e) {
            System.out.println(e.getMessage());
        } catch (InvalidRegistrationNumberException e) {
            System.out.println(e.getMessage());
        } catch (Exception e) {
            System.out.println("Incorrect Data type entered");
        }
    }

}
```

Output









Question8

Write a JAVA program to show the usage of throws keyword with user defined exceptions

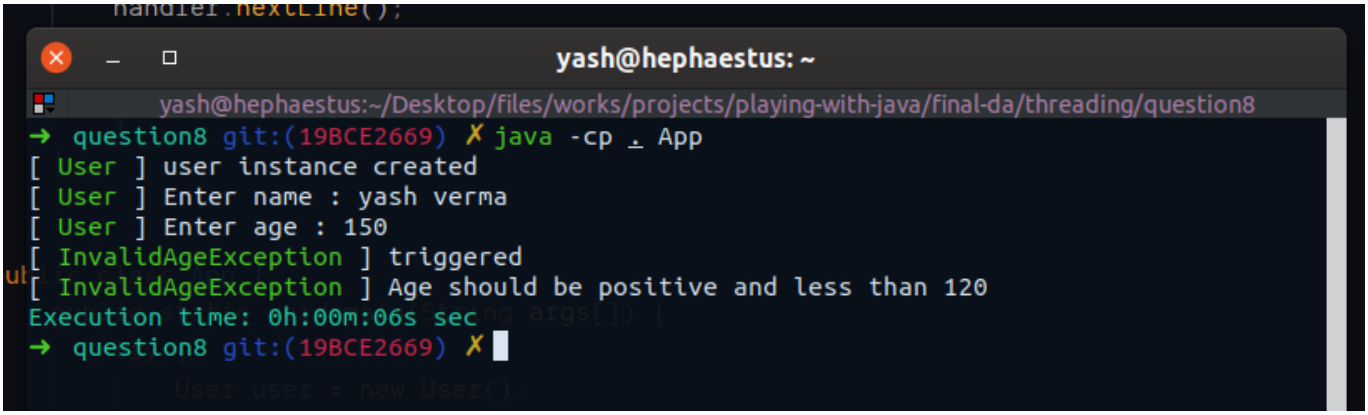
Code

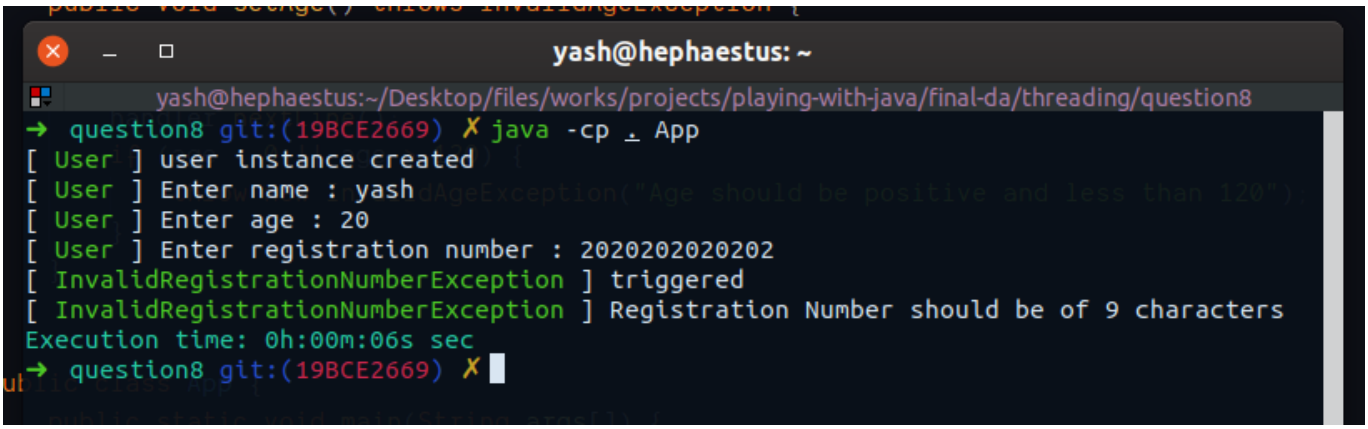
```
import java.util.InputMismatchException;
import java.util.Scanner;
import rainbow.rainbow;

class InvalidNameException extends Exception {
    private final String context = "[ " +
rainbow.green("InvalidNameException") + " ] ";
    private String message;

    public InvalidNameException(String message) {
        super(message);
        this.message = message;
        System.out.println(this.context + "triggered");
    }

    public String getMessage() {
        return this.context + this.message;
    }
};

class InvalidAgeException extends Exception {
    private final String context = "[ " +
```

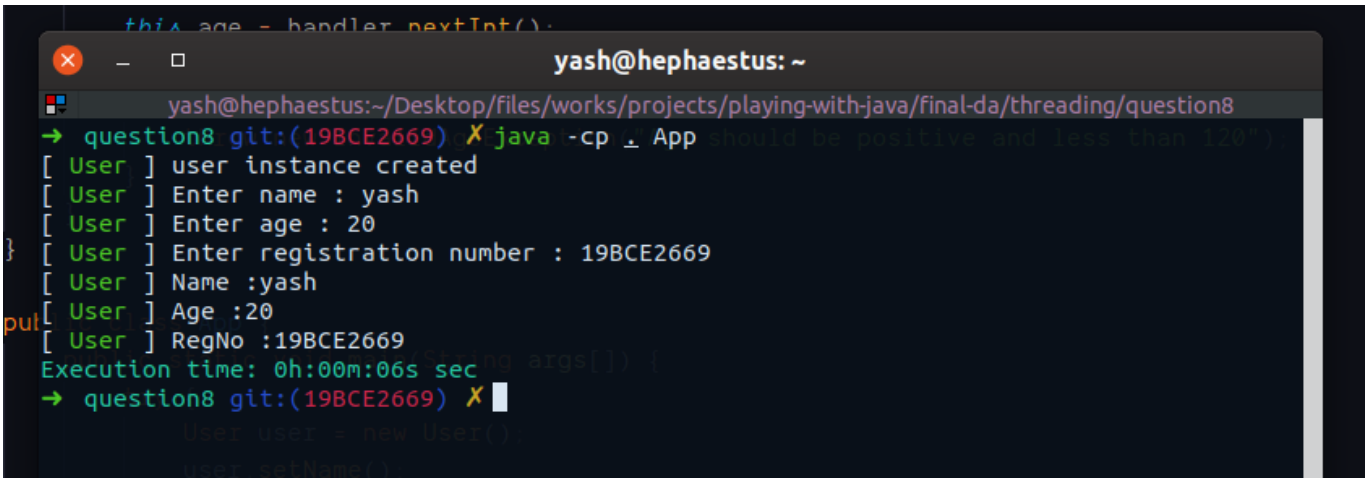
```
rainbow.green("InvalidAgeException") + " ] ";
    private String message;

    public InvalidAgeException(String message) {
        super(message);
        this.message = message;
        System.out.println(this.context + "triggered");
    }

    public String getMessage() {
        return this.context + this.message;
    }
};

class InvalidRegistrationNumberException extends Exception {
    private final String context = "[ " +
rainbow.green("InvalidRegistrationNumberException") + " ] ";
    private String message;

    public InvalidRegistrationNumberException(String message) {
        super(message);
        this.message = message;
        System.out.println(this.context + "triggered");
    }

    public String getMessage() {
        return this.context + this.message;
    }
};

class User {
    final private String context = "[ " + rainbow.green("User") + " ] ";
    private String name;
    private String regNo;
    private int age;
    private Scanner handler = new Scanner(System.in);

    User() {
        System.out.println(this.context + "user instance created");
    }

    public void displayDetails() {
        System.out.println(this.context + "Name :" + this.name);
        System.out.println(this.context + "Age :" + this.age);
        System.out.println(this.context + "RegNo :" + this.regNo);
    }

    public void setName() throws InvalidNameException {
        System.out.print(context + "Enter name : ");
        this.name = handler.nextLine();
        if (name.length() <= 3) {
            throw new InvalidNameException("Name should be longer than 3
characters");
        }
```

```
    }

    public void setRegNo() throws InvalidRegistrationNumberException {
        System.out.print(context + "Enter registration number : ");
        this.regNo = handler.nextLine();
        if (regNo.length() != 9) {
            throw new InvalidRegistrationNumberException("Registration
Number should be of 9 characters");
        }
    }

    public void setAge() throws InvalidAgeException {
        System.out.print(context + "Enter age : ");
        this.age = handler.nextInt();
        handler.nextLine();
        if (age < 0 || age > 120) {
            throw new InvalidAgeException("Age should be positive and less
than 120");
        }
    }
}

public class App {
    public static void main(String args[]) {
        try {
            User user = new User();
            user.setName();
            user.setAge();
            user.setRegNo();
            user.displayDetails();
        } catch (InvalidNameException e) {
            System.out.println(e.getMessage());
        } catch (InvalidAgeException e) {
            System.out.println(e.getMessage());
        } catch (InvalidRegistrationNumberException e) {
            System.out.println(e.getMessage());
        } catch (Exception e) {
            System.out.println("Incorrect Data type entered");
        }
    }

}
```

Output







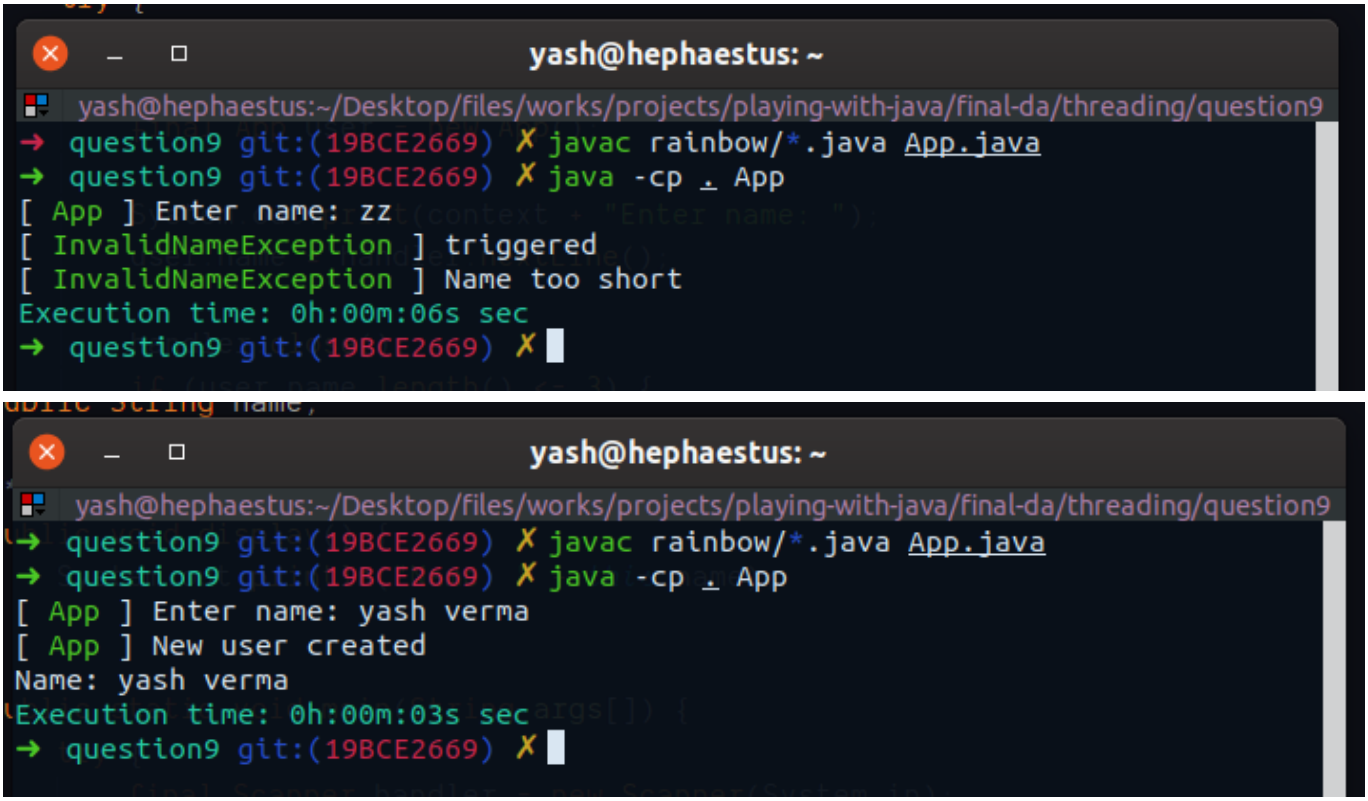
Question9

Write a JAVA program to throw a custom exception without inheriting exceptions class.

Code

```
import java.util.Scanner;
import java.util.logging.Handler;
import rainbow.rainbow;

/** extending throwable as custom Exception */
class InvalidNameException extends Throwable {
    private final String context = "[ " +
rainbow.green("InvalidNameException") + " ] ";
    private String message;

    InvalidNameException(String message) {
        super(message);
        this.message = message;
        System.out.println(this.context + "triggered");
    }

    public String getMessage() {
        return this.context + this.message;
    }
}

/** main caller class */
public class App {
    public String name;

    /** display details of user after created */
    public void display() {
        System.out.println("Name: " + this.name);
    }

    public static void main(String args[]) {
        try {
            final Scanner handler = new Scanner(System.in);
            final String context = "[ " + rainbow.green("App") + " ] ";
            final App user = new App();

            System.out.print(context + "Enter name: ");
            user.name = handler.nextLine();

            handler.close();
            if (user.name.length() <= 3) {
                throw new InvalidNameException("Name too short");
            } else {
                System.out.println(context + "New user created");
                user.display();
            }
        } catch (InvalidNameException e) {
            System.out.println(e.getMessage());
        }

    }
}
```

Output



Question10

Write a JAVA program to demonstrate usage of finally block.

Code

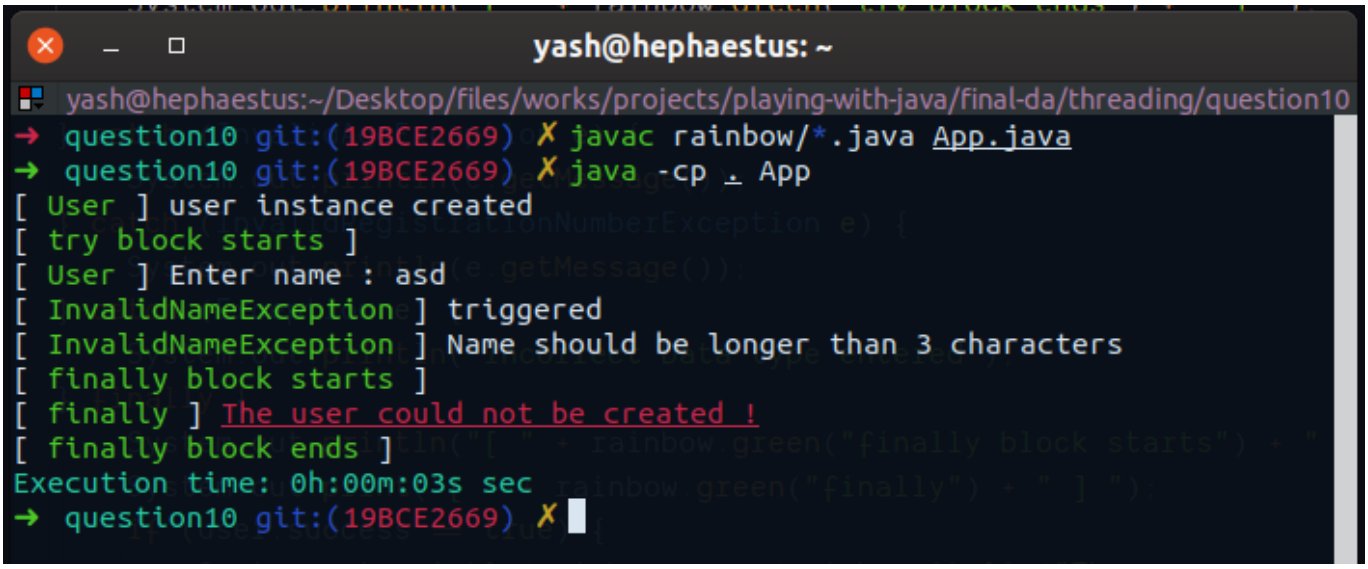
```
import java.util.InputMismatchException;
import java.util.Scanner;
import rainbow.rainbow;

class InvalidNameException extends Exception {
    private final String context = "[ " +
rainbow.green("InvalidNameException") + " ] ";
    private String message;

    public InvalidNameException(String message) {
        super(message);
        this.message = message;
        System.out.println(this.context + "triggered");
    }

    public String getMessage() {
        return this.context + this.message;
    }
};

class InvalidAgeException extends Exception {
```

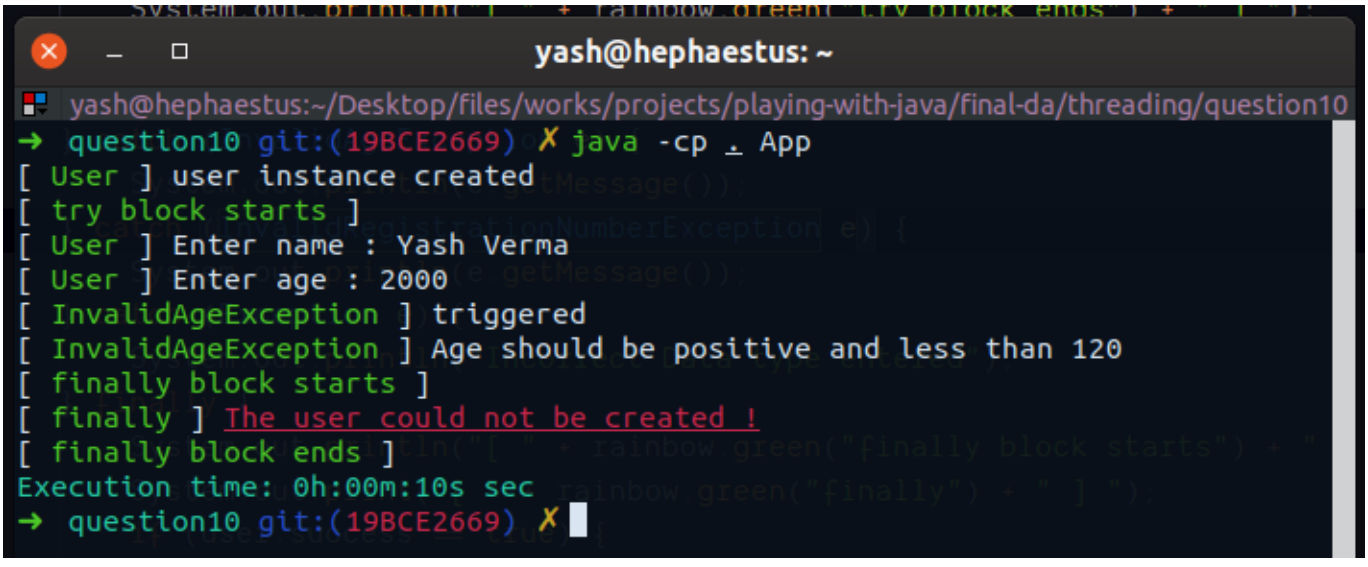
```
    private final String context = "[ " +
rainbow.green("InvalidAgeException") + " ] ";
    private String message;

    public InvalidAgeException(String message) {
        super(message);
        this.message = message;
        System.out.println(this.context + "triggered");
    }

    public String getMessage() {
        return this.context + this.message;
    }
};

class InvalidRegistrationNumberException extends Exception {
    private final String context = "[ " +
rainbow.green("InvalidRegistrationNumberException") + " ] ";
    private String message;

    public InvalidRegistrationNumberException(String message) {
        super(message);
        this.message = message;
        System.out.println(this.context + "triggered");
    }

    public String getMessage() {
        return this.context + this.message;
    }
};

class User {
    final private String context = "[ " + rainbow.green("User") + " ] ";
    private String name;
    private String regNo;
    private int age;
    private Scanner handler = new Scanner(System.in);
    public Boolean success = false;

    User() {
        System.out.println(this.context + "user instance created");
    }

    public void displayDetails() {
        System.out.println(this.context + "User Details:");
        System.out.println(this.context + "Name :" + this.name);
        System.out.println(this.context + "Age :" + this.age);
        System.out.println(this.context + "RegNo :" + this.regNo);
    }

    public void setName() throws InvalidNameException {
        System.out.print(context + "Enter name : ");
        this.name = handler.nextLine();
        if (name.length() <= 3) {
```

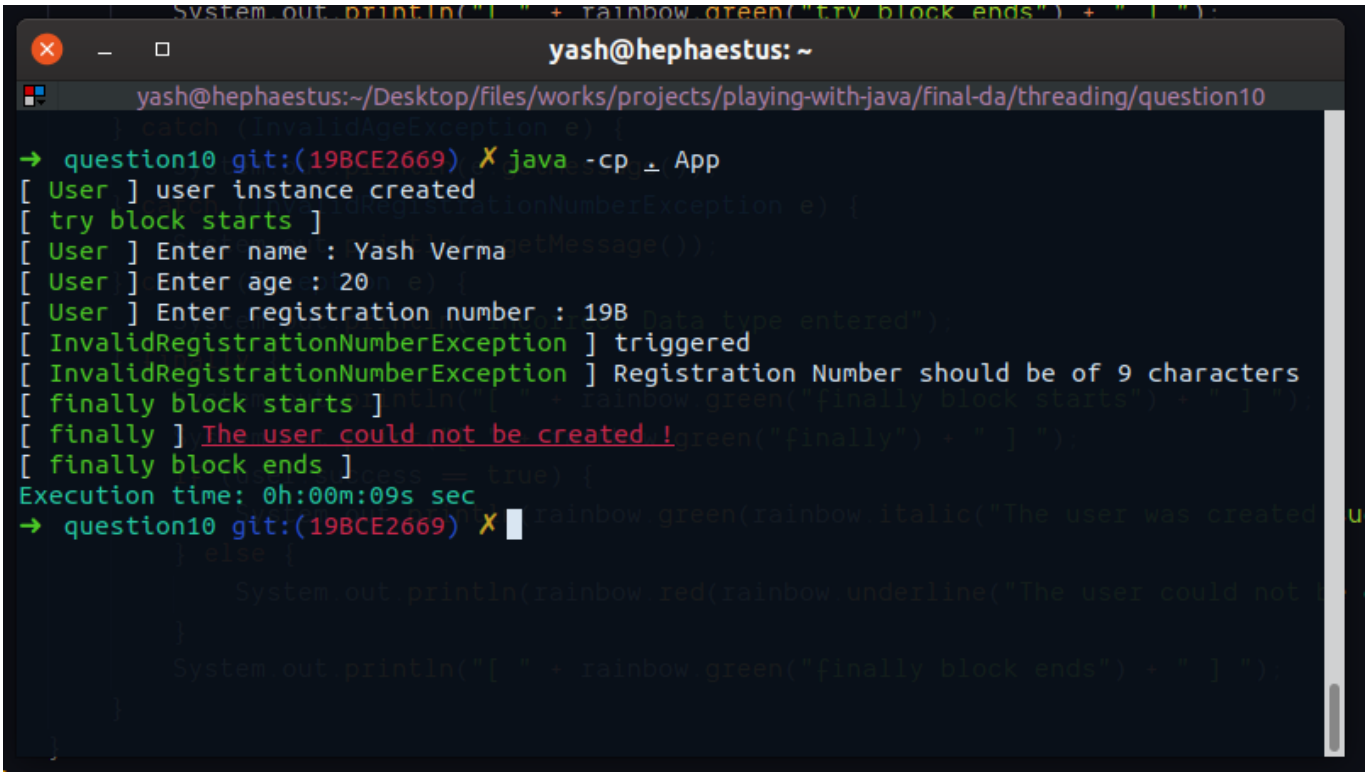
```
            throw new InvalidNameException("Name should be longer than 3
characters");
        }
    }

    public void setRegNo() throws InvalidRegistrationNumberException {
        System.out.print(context + "Enter registration number : ");
        this.regNo = handler.nextLine();
        if (regNo.length() != 9) {
            throw new InvalidRegistrationNumberException("Registration
Number should be of 9 characters");
        }
    }

    public void setAge() throws InvalidAgeException {
        System.out.print(context + "Enter age : ");
        this.age = handler.nextInt();
        handler.nextLine();
        if (age < 0 || age > 120) {
            throw new InvalidAgeException("Age should be positive and less
than 120");
        }
    }
}

public class App {
    public static void main(String args[]) {

        User user = new User();
        try {

            System.out.println("[ " + rainbow.green("try block starts") + "
] ");
            user.setName();
            user.setAge();
            user.setRegNo();
            user.success = true;
            System.out.println("[ " + rainbow.green("try block ends") + " ]
");
        } catch (InvalidNameException e) {
            System.out.println(e.getMessage());
        } catch (InvalidAgeException e) {
            System.out.println(e.getMessage());
        } catch (InvalidRegistrationNumberException e) {
            System.out.println(e.getMessage());
        } catch (Exception e) {
            System.out.println("Incorrect Data type entered");
        } finally {
            System.out.println("[ " + rainbow.green("finally block starts")
+ " ] ");
            System.out.print("[ " + rainbow.green("finally") + " ] ");
            if (user.success == true) {
                System.out.println(rainbow.green(rainbow.underline("The
user was created successfully.")));
```

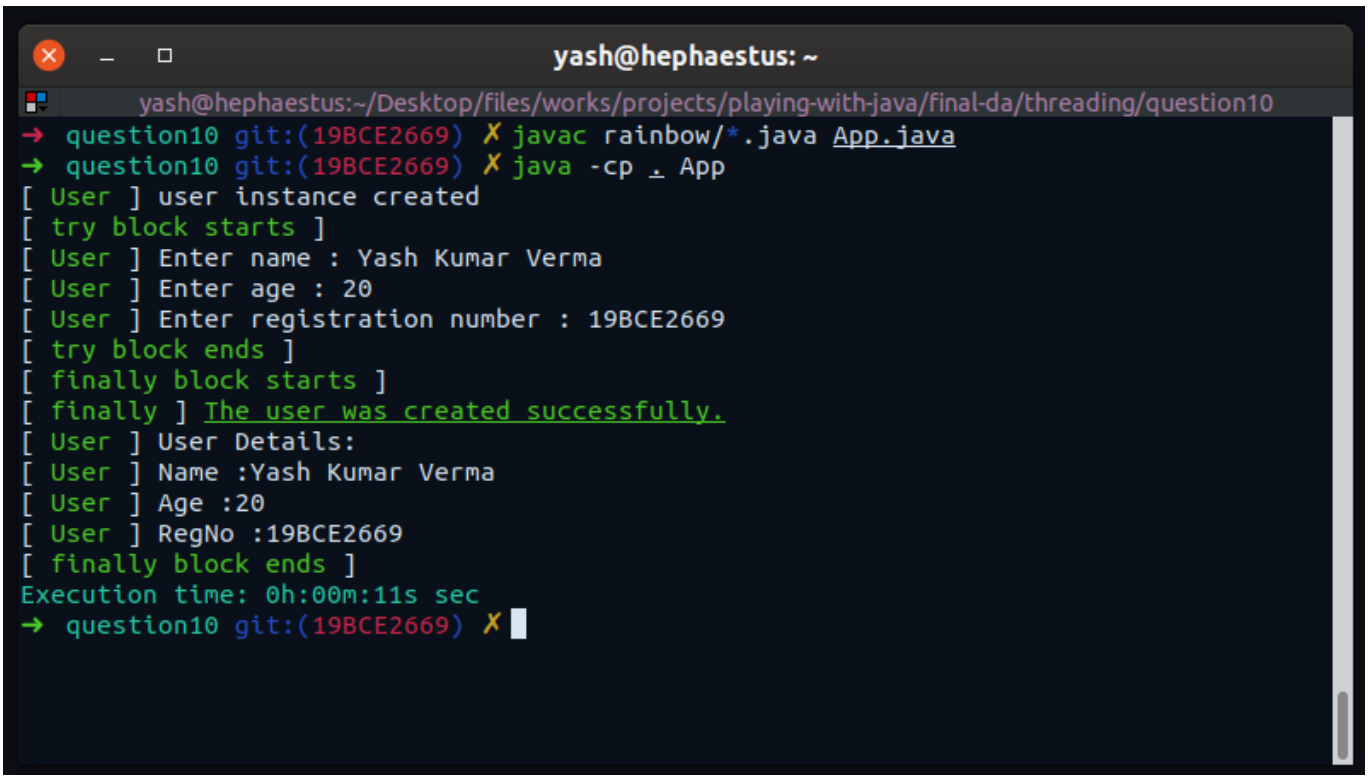
```
            } else {
                System.out.println(rainbow.red(rainbow.underline("The user
  could not be created !")));
            }
            user.displayDetails();
            System.out.println("[ " + rainbow.green("finally block ends") +
  " ] ");
        }
    }

}
```

Output






yash@hephaestus: ~
yash@hephaestus:~/Desktop/files/works/projects/playing-with-java/final-da/threading/question10
➜ question10 git:(19BCE2669) ✗ java -cp . App
[User] user instance created
[try block starts]
[User] Enter name : Yash Verma
[User] Enter age : 20
[User] Enter registration number : 19B
[InvalidRegistrationNumberException] triggered
[InvalidRegistrationNumberException] Registration Number should be of 9 characters
[finally block starts]
[finally] The user could not be created !
[finally block ends]
Execution time: 0h:00m:09s sec
➜ question10 git:(19BCE2669) ✗


yash@hephaestus: ~
yash@hephaestus:~/Desktop/files/works/projects/playing-with-java/final-da/threading/question10
➜ question10 git:(19BCE2669) ✗ javac rainbow/*.java App.java
➜ question10 git:(19BCE2669) ✗ java -cp . App
[User] user instance created
[try block starts]
[User] Enter name : Yash Kumar Verma
[User] Enter age : 20
[User] Enter registration number : 19BCE2669
[try block ends]
[finally block starts]
[finally] The user was created successfully.
[User] User Details:
[User] Name :Yash Kumar Verma
[User] Age :20
[User] RegNo :19BCE2669
[finally block ends]
Execution time: 0h:00m:11s sec
➜ question10 git:(19BCE2669) ✗