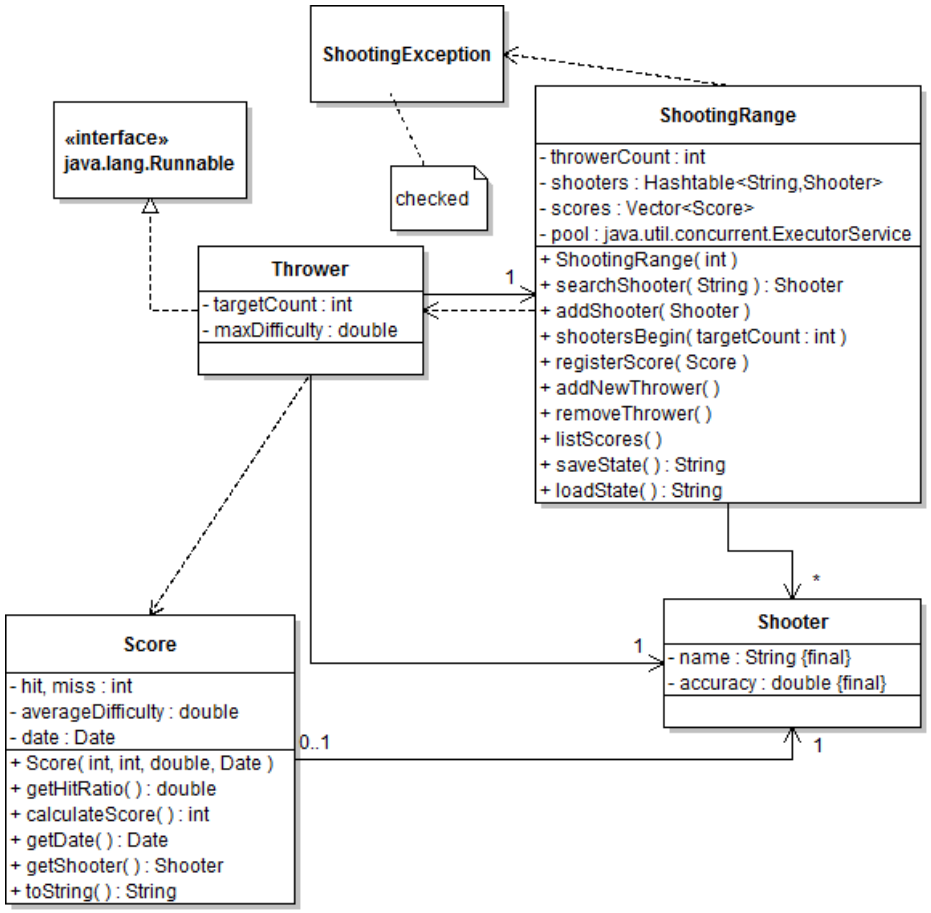
BLM2562 OBJECT ORIENTED PROG., Gr.2 & 3, 2nd MIDTERM (11/05/2016)

Duration:	90 mins.			**Score:**				**Student Nr:**	**Signature:**
Grading:	**1**	**2**	**3**	**4**	**5**	**6**	**Group**	**Name, Surname:**	
	10	10	40	40					

QUESTIONS



Domain information:

Shooters make target practice in a shooting range. There must be at least one thrower in a shooting range. Each shooter practices in one thrower. If a shooter cannot find an empty thrower, s/he waits until one becomes empty.

Answer the following questions according to the UML class schema given above. You may need to extract hidden information from the schema and add necessary code while answering the questions.

Question 1: Write the source code of class Shooter.
Question 2: Write the source code of ShootingException.
Question 3: Write the source code of class Thrower. An instance of this multithreaded class makes its shooter to shoot for targetCount times and prints the result of each shot to the console. It generates a random double number between 0 and 1 for each shot. That random number stands for the difficulty of one target. If the accuracy of the shooter is greater than that difficulty, this shot is a hit. Otherwise, the shot is a miss. A random time between 0 and 1 seconds must pass between each shot. After all shots are done, the score must be registered into the related ShootingRange instance.
Question 4: Write the source code of the following methods of class ShootingRange. Don't forget to handle multithreading issues.

- addShooter: No two shooters having the same name exist. If one attempts to do so, an exception with detailed information must be generated.
- shootersBegin: All shooters begin shooting. P.S. Use Thrower instances.
- registerScore: A score, generated at the end of a Thrower thread, is registered within the Vector.
- saveState: Carry out the task. You must wait for the shooters to end their shootings before saving.

BLM2562 OBJECT ORIENTED PROG., Gr.2 & 3, 2nd MIDTERM (11/05/2016)

Question 1: Write the source code of class Shooter.

```
public class Shooter implements java.io.Serializable {
	private final String name;
	private final double accuracy;

	public Shooter(String name, double accuracy) {
		this.name = name; this.accuracy = accuracy;
	}
	public String getName() { return name; }
	public double getAccuracy() { return accuracy; }

	public String toString( ) {
		return name + " (%" + (accuracy * 100) + ")";
	}
}
```

Question 2: Write the source code of ShootingException.

```
public class ShootingException extends IOException {
	public ShootingException( String msg ) {
		super(msg);
	}
}
```

Question 3: Write the source code of class Thrower.

```
import java.util.*;
public class Thrower implements Runnable {
	private int targetCount, hitCount;
	private Shooter shooter;
	private ShootingRange range;
	public Thrower(int targetCount, Shooter shooter,
			ShootingRange range) {
		this.targetCount = targetCount; this.shooter = shooter;
		this.range = range; hitCount = 0;
	}
	public void run() {
		Random generator = new Random();
		double difficulty, total = 0.0;
		try {
			for( int i=0; i<targetCount;i++ ) {
				difficulty = generator.nextDouble();
				total += difficulty;
				if( shooter.getAccuracy() > difficulty ) {
					hitCount++;
					System.out.println(shooter + " hits target @"
							+ String.format("%.3f", difficulty));
				}
				else {
					System.out.println(shooter + " misses target @"
							+ String.format("%.3f", difficulty));
				}
				Thread.sleep( generator.nextInt(1000) );
			}
		}
		catch (InterruptedException e) {
			e.printStackTrace();
		}
		Score score = new Score(hitCount,(targetCount-hitCount),
				total/hitCount,new Date(),shooter);
		range.registerScore(score);
	}
}
```

Question 4: Write the source code of the following methods of class ShootingRange.

```
	public void addShooter( Shooter shooter ) throws ShootingException {
		if( searchShooter(shooter.getName()) != null )
			throw new ShootingException("Duplicate shooter: " + shooter.getName());
		shooters.put(shooter.getName(), shooter);
	}
	public void shootersBegin( int targetCount ) {
		for( Shooter shooter : shooters.values() )
			pool.execute( new Thrower( targetCount, shooter, this ) );
		pool.shutdown();
	}
	public synchronized void registerScore( Score newScore ) {
		scores.add(newScore);
	}
	public void saveState( String file ) {
		while( !pool.isTerminated() );
		try {
			ObjectOutputStream stream = new ObjectOutputStream(
					new FileOutputStream(file) );
			stream.writeObject(throwerCount);
			stream.writeObject(shooters);
			stream.writeObject(scores);
			stream.close();
		}
		catch (IOException e) {
			e.printStackTrace();
		}
	}
```

Meraklısına tüm kodlar:

```
import java.util.*;
@SuppressWarnings("serial")
public class Score implements java.io.Serializable {
	private int hit, miss;
	private double averageDifficulty;
	private Date date;
	private Shooter shooter;

	public Score(int hit, int miss, double avrg,
			Date date, Shooter shooter) {
		this.hit = hit; this.miss = miss;
		averageDifficulty = avrg; this.date = date;
		this.shooter = shooter;
	}
	public double getHitRatio( ) {
		return (double)hit/(hit+miss);
	}
	public int calculateScore( ) {
		return (int) (getHitRatio() * averageDifficulty * 100);
	}
	public Date getDate() { return date; }
	public Shooter getShooter() { return shooter; }
	public String toString() {
		return "Hit ratio: " + String.format( "%.2f", getHitRatio( ) )
				+ " by "+ shooter.getName()
				+ "; Score: " + calculateScore( )
				+ " at " + date;
	}
}
```

```
@SuppressWarnings("serial")
public class Shooter implements java.io.Serializable {
	private final String name;
	private final double accuracy;

	public Shooter(String name, double accuracy) {
		this.name = name; this.accuracy = accuracy;
	}
	public String getName() { return name; }
	public double getAccuracy() { return accuracy; }

	public String toString( ) {
		return name + " (%" + (accuracy * 100) + ")";
	}
}

import java.util.*;
import java.util.concurrent.*;
import java.io.*;

public class ShootingRange {
	private Integer throwerCount;
	private Hashtable<String,Shooter> shooters;
	private Vector<Score> scores;
	private ExecutorService pool;

	public ShootingRange( int throwerCount ) {
		this.throwerCount = throwerCount;
		shooters = new Hashtable<String,Shooter>();
		scores = new Vector<Score>();
		pool = Executors.newFixedThreadPool(throwerCount);
	}
	public Shooter searchShooter( String name ) {
		return shooters.get(name);
	}
	public void addShooter( Shooter shooter ) throws ShootingException {
		if( searchShooter(shooter.getName()) != null )
			throw new ShootingException("Duplicate shooter: " + shooter.getName());
		shooters.put(shooter.getName(), shooter);
	}
	public void shootersBegin( int targetCount ) {
		for( Shooter shooter : shooters.values() )
			pool.execute( new Thrower( targetCount, shooter, this ) );
		pool.shutdown();
	}
	public synchronized void registerScore( Score newScore ) {
		scores.add(newScore);
	}
	public void addNewThrower( ) {
		while( !pool.isTerminated() );
		throwerCount++;
		pool = Executors.newFixedThreadPool(throwerCount);
	}
	public void removeThrower( ) throws ShootingException {
		if( throwerCount == 1 )
			throw new ShootingException("Cannot remove the last thrower.");
		while( !pool.isTerminated() );
		throwerCount--;
		pool = Executors.newFixedThreadPool(throwerCount);
	}
	public void listScores( ) {
		while( !pool.isTerminated() );
		for( Score score : scores )
			System.out.println(score);
	}
```

```
	public void saveState( String file ) {
		while( !pool.isTerminated() );
		try {
			ObjectOutputStream stream = new ObjectOutputStream(
					new FileOutputStream(file) );
			stream.writeObject(throwerCount);
			stream.writeObject(shooters);
			stream.writeObject(scores);
			stream.close();
		}
		catch (IOException e) {
			e.printStackTrace();
		}
	}
	@SuppressWarnings("unchecked")
	public void loadState( String file ) {
		while( !pool.isTerminated() );
		try {
			ObjectInputStream stream = new ObjectInputStream(
					new FileInputStream(file) );
			throwerCount = (Integer)stream.readObject();
			shooters = (Hashtable<String,Shooter>)stream.readObject();
			scores = (Vector<Score>)stream.readObject();
			stream.close();
		}
		catch (IOException e) {
			e.printStackTrace();
		}
		catch (ClassNotFoundException e) {
			e.printStackTrace();
		}
	}
}

public class MainProgram {
	public static void main(String[] args) {
		try {
			Shooter sniper1 = new Shooter("Vasily Zaytsev", 0.95);
			Shooter sniper2 = new Shooter("Erwin König", 0.90);
			Shooter rookie = new Shooter("Yunus Emre Selçuk", 0.65);
			ShootingRange range = new ShootingRange(2);
			range.addShooter(sniper1);
			range.addShooter(sniper2);
			range.addShooter(rookie);
			range.shootersBegin(4);
			range.listScores();
			range.saveState("araSinavTest.dat");
		}
		catch (ShootingException e) {
			e.printStackTrace();
		}
	}

}
```