Faculty of Computers & Information
Distributed Systems
Instructor: **Dr. Anas Youssef**

جامعة المنوفية

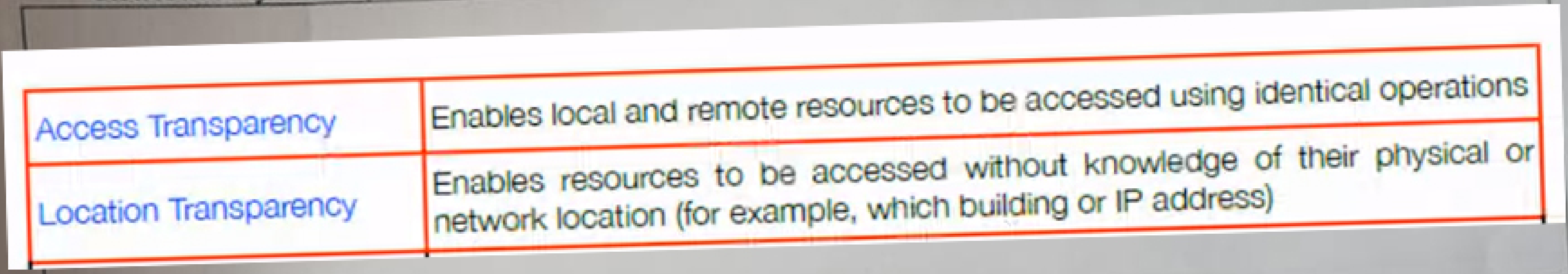
Midterm Exam
Time: **60 minutes**
Number of Pages: **4**
Total marks: **20**

Student Name	فريد محمد فريد نوار
Student Section Number	

Question 1:

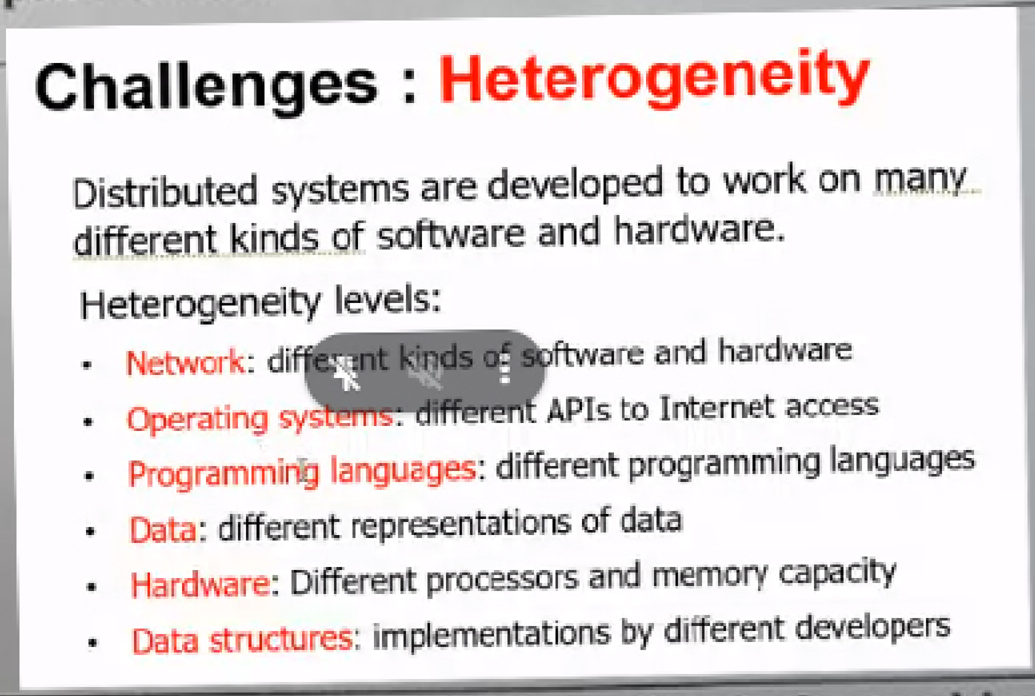
(6 Marks)

(a) Briefly describe the difference between **access transparency** and **location transparency** while characterizing distributed systems. (2 marks)



Access Transparency	Enables local and remote resources to be accessed using identical operations
Location Transparency	Enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address)

(b) List **three different examples of heterogenous components** that can be located on a server and a corresponding client machine. (2 marks)



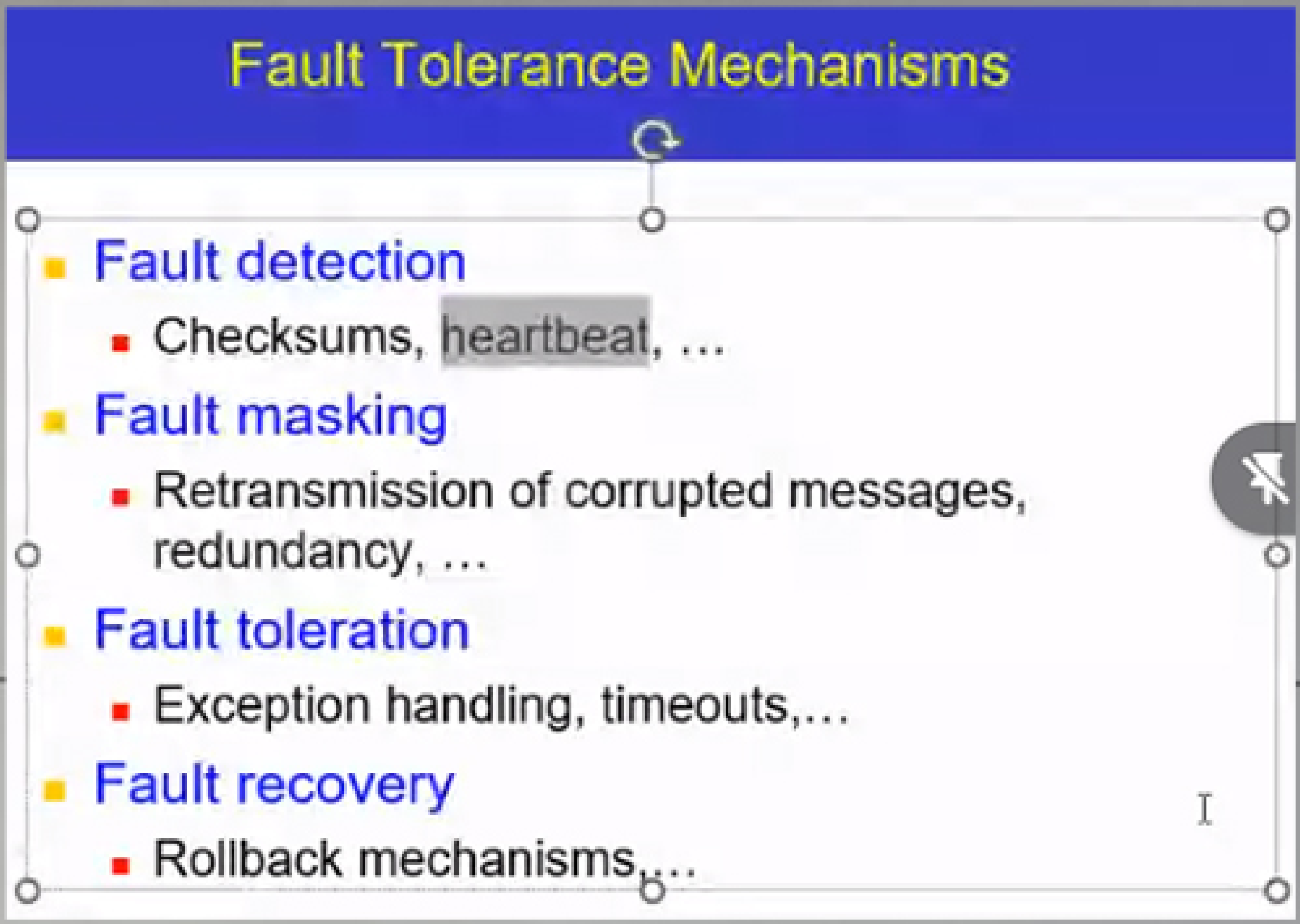
Challenges : Heterogeneity

Distributed systems are developed to work on many different kinds of software and hardware.

Heterogeneity levels:

- Network: different kinds of software and hardware
- Operating systems: different APIs to Internet access
- Programming languages: different programming languages
- Data: different representations of data
- Hardware: Different processors and memory capacity
- Data structures: implementations by different developers

(c) Briefly describe **one** technique used for **fault detection** and **another** technique for **fault masking** in distributed systems. (2 marks)



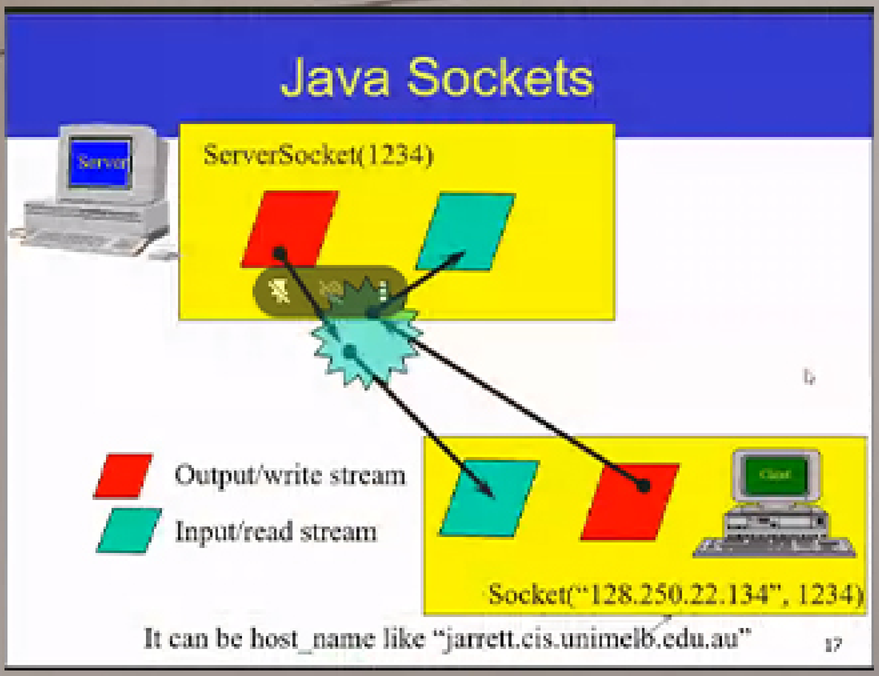
Fault Tolerance Mechanisms

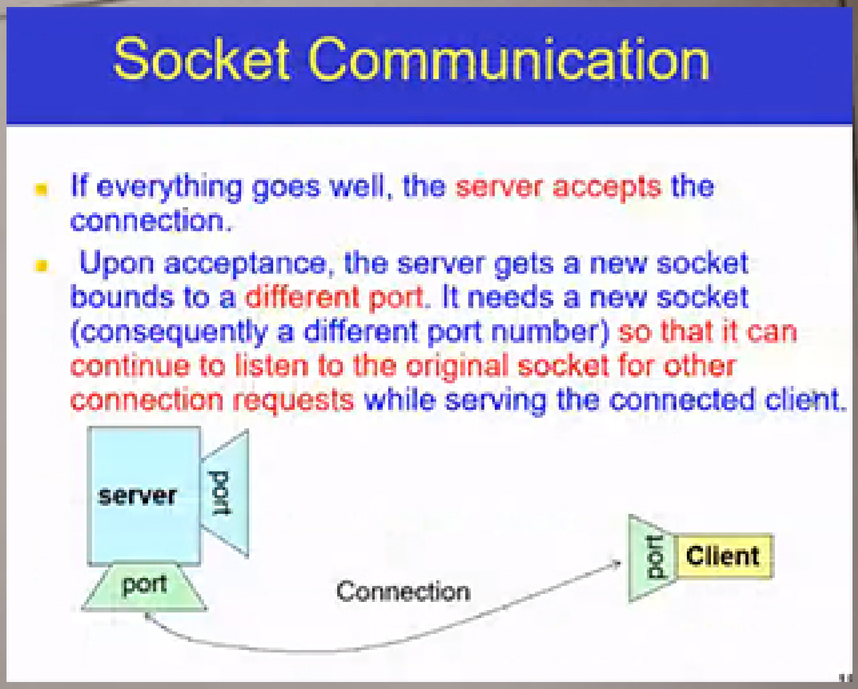
- Fault detection
  - Checksums, heartbeat, …
- Fault masking
  - Retransmission of corrupted messages, redundancy, …
- Fault toleration
  - Exception handling, timeouts,…
- Fault recovery
  - Rollback mechanisms,…

(9 Marks)

Question 2:

(a) Suppose that a program runs on a specific server, and there are two client machines who would like to communicate with that program. **Sketch a diagram** to show the complete sequence of communication, through several communication requests, between the two clients and that program using the **TCP protocol**. Your diagram needs to clearly show the detailed IP address(es), port number(s), socket(s) as needed. (4 marks)




Socket Communication

- If everything goes well, the server accepts the connection.
- Upon acceptance, the server gets a new socket bounds to a different port. It needs a new socket (consequently a different port number) so that it can continue to listen to the original socket for other connection requests while serving the connected client.

server
port
port
Connection
port
Client

✗ (b) What is meant by an **idempotent operation** when executed by a server? Provide an **example** for one operation that fits as an idempotent operation, and another example for an operation that does not fit as an idempotent operation. **Justify why** you see each operation fits within its classification. (3 marks)

(c) Which of the following three types of thread architectures can provide more thread parallelism at the server: **Thread per request**, **Thread per connection** or **Thread per remote object**? **Why**? (2 marks)

"Thread per request" architecture offers more thread parallelism at the server as it allows concurrent processing of multiple independent requests.

(5 Marks)

Question 3:

(a) Consider the following Java program that involves a **CookieIssuer** class that issues cookies, a **CookieMonster** class that eats cookies, and a **CookiePlate** class. **Modify** the program such that you don't need to extend the Thread class in the CookieIssuer and CookieMonster classes to support multiple threads. However, threading should still be supported in the code. (3 marks)

```
public class CookiesAppTest {
        public static void main(String[] args) {
                CookiePlate c = new CookiePlate();
                CookieIssuer p1 = new CookieIssuer(c, 1);
                CookieMonster c1 = new CookieMonster(c, 1);
                p1.start();
                c1.start();
        }
}
```

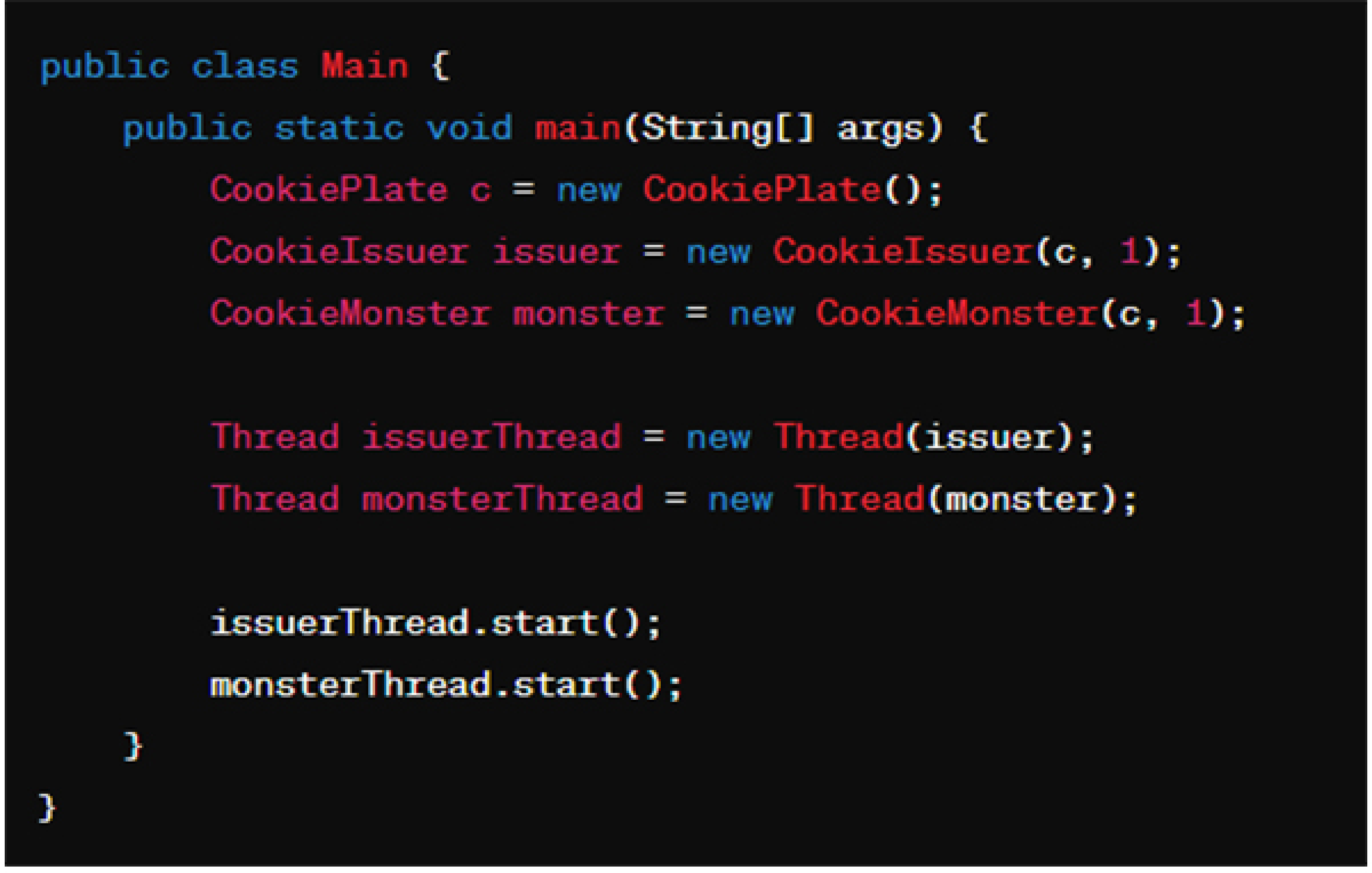
```
public class CookieIssuer extends Thread{
        private CookiePlate cookiePlate;
        private int number;
        public CookieIssuer
                (CookiePlate c, int   number) {
                cookiePlate = c;
                this.number = number;
        }
```

```
public void run() {
 for (int i = 0; i < 10; i++) {
     cookiePlate.put(i);
     System.out.println("Monster #" +
this.number + " put: " + i);
     try {
        sleep((int)(Math.random() * 100));
     } catch (InterruptedException e) { }
   }
 }
}
```

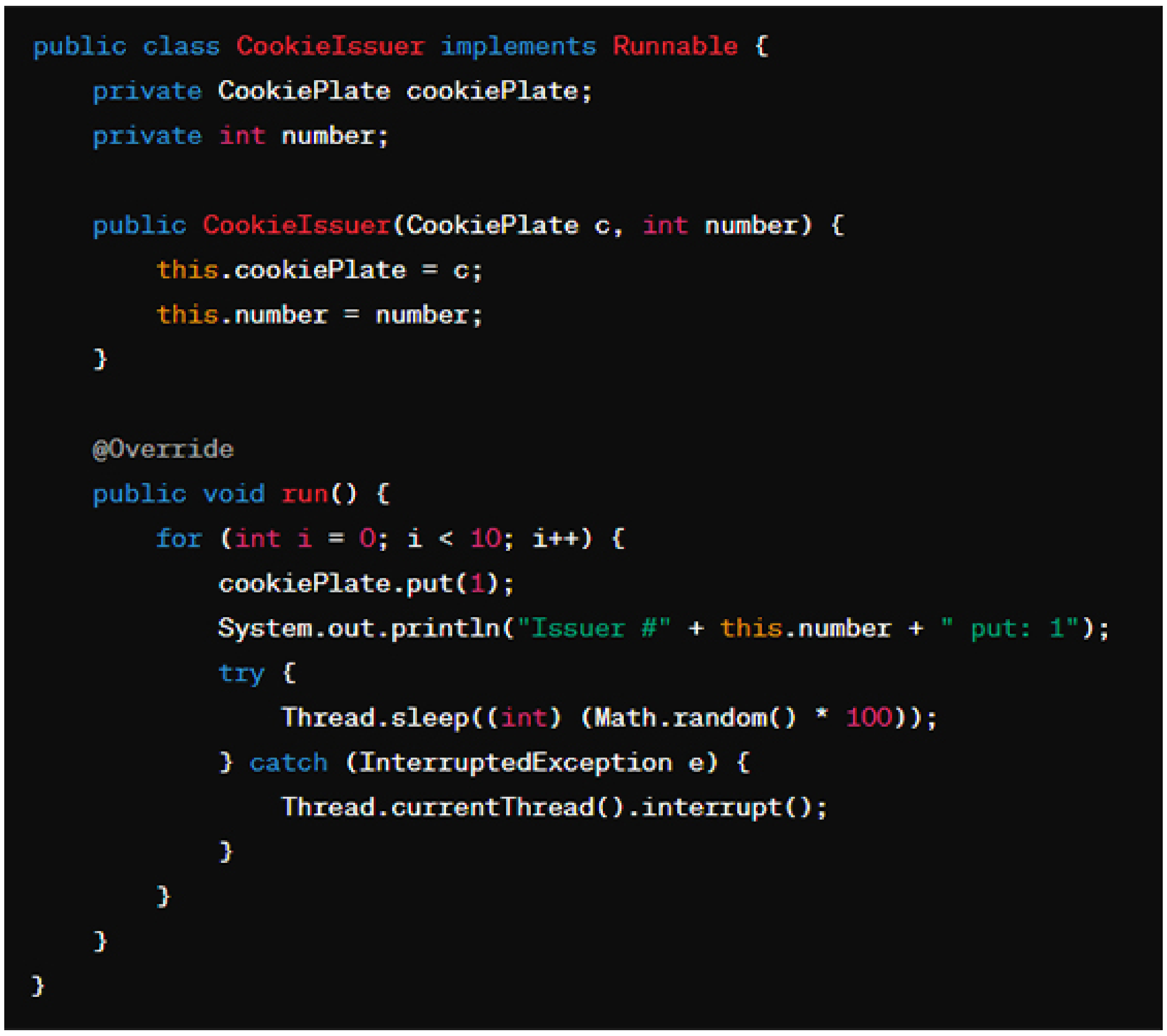
```
public class CookieMonster extends Thread{

        private CookiePlate cookiePlate;
        private int number;
        public CookieMonster
                (CookiePlate c, int number) {
                cookiePlate = c;
                this.number = number;
        }
```

```
public void run() {
  for ( int i=0; i<10; i++) {
      System.out.println ("CookieMonster# "+
this.number + "get: "+ cookiePlate.get());
      try {
         sleep ((int)(Math.random()*100));
      } catch (InterruptedException e) {}
   }
 }
}
```

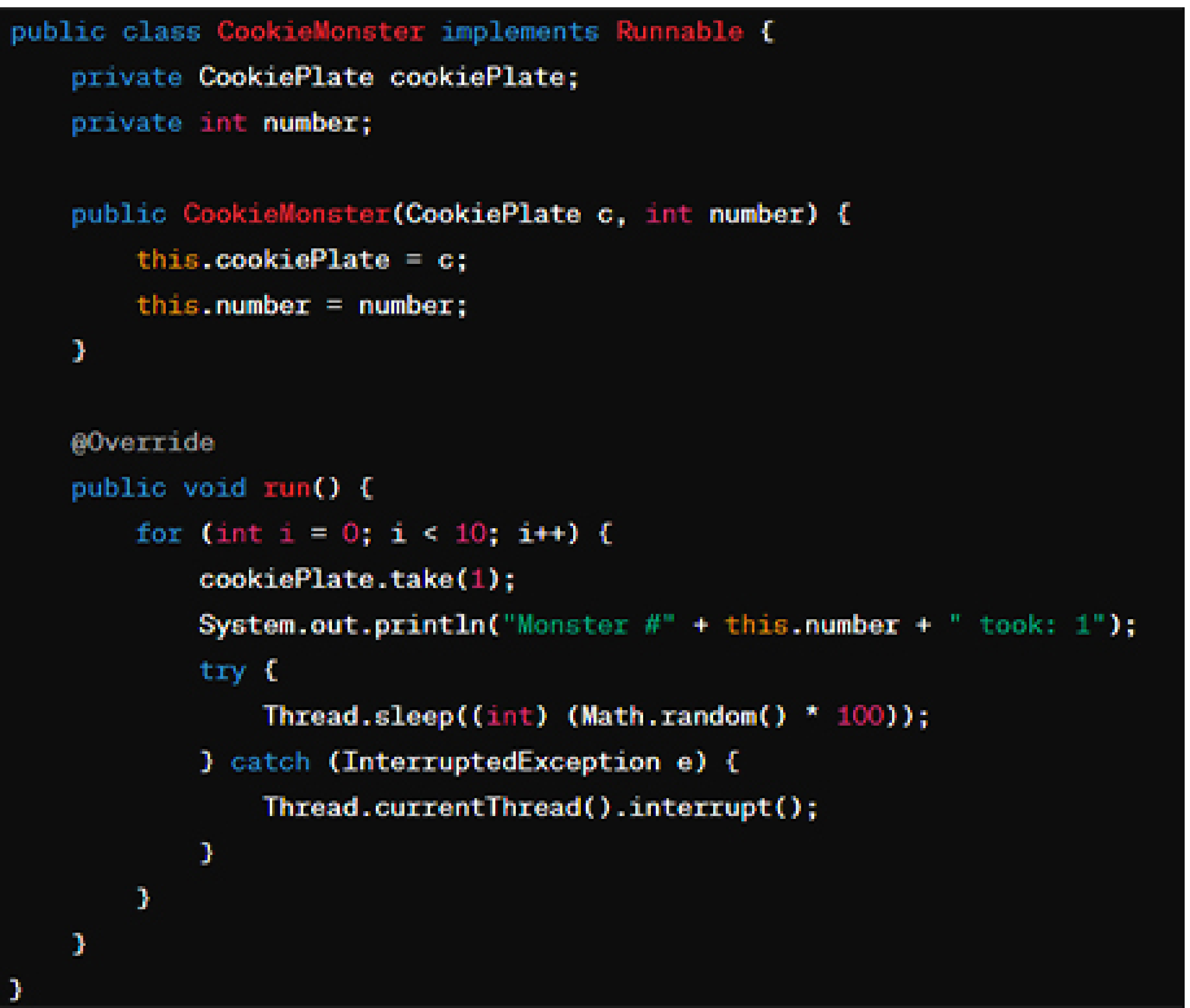
```
public class CookiePlate {

  private int contents;
  private boolean available = false;
  public int get() {
   //some code to get a value from the plate
  }
   public void put(int value) {
  //some code to put a value in the plate
}
```



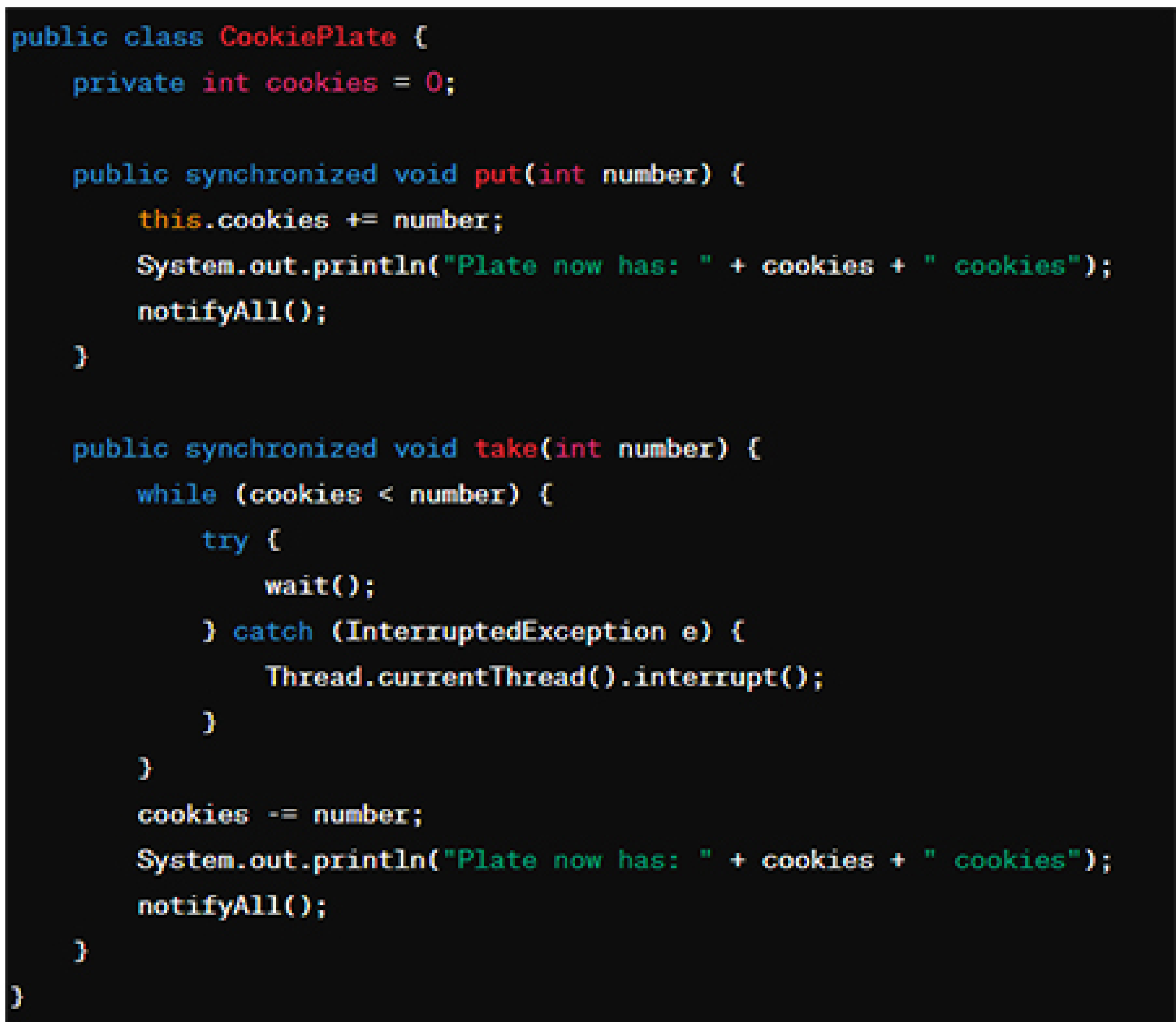
```java
public class Main {
    public static void main(String[] args) {
        CookiePlate c = new CookiePlate();
        CookieIssuer issuer = new CookieIssuer(c, 1);
        CookieMonster monster = new CookieMonster(c, 1);

        Thread issuerThread = new Thread(issuer);
        Thread monsterThread = new Thread(monster);

        issuerThread.start();
        monsterThread.start();
    }
}
```



```java
public class CookieIssuer implements Runnable {
    private CookiePlate cookiePlate;
    private int number;

    public CookieIssuer(CookiePlate c, int number) {
        this.cookiePlate = c;
        this.number = number;
    }

    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            cookiePlate.put(1);
            System.out.println("Issuer #" + this.number + " put: 1");
            try {
                Thread.sleep((int) (Math.random() * 100));
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
```



```java
public class CookieMonster implements Runnable {
    private CookiePlate cookiePlate;
    private int number;

    public CookieMonster(CookiePlate c, int number) {
        this.cookiePlate = c;
        this.number = number;
    }

    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            cookiePlate.take(1);
            System.out.println("Monster #" + this.number + " took: 1");
            try {
                Thread.sleep((int) (Math.random() * 100));
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
```



```java
public class CookiePlate {
    private int cookies = 0;

    public synchronized void put(int number) {
        this.cookies += number;
        System.out.println("Plate now has: " + cookies + " cookies");
        notifyAll();
    }

    public synchronized void take(int number) {
        while (cookies < number) {
            try {
                wait();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        cookies -= number;
        System.out.println("Plate now has: " + cookies + " cookies");
        notifyAll();
    }
}
```