

Basic Thread Synchronization

**Programación Concurrente
2017**

Ing. Ventre, Luis O.

Basic Thread Synchronization

- Una de las situaciones mas comunes en programas concurrentes ocurre cuando más de un hilo en ejecución comparten un recurso.
 - En aplicaciones concurrentes es común que varios hilos escriban y lean los mismos recursos.
 - Estos recursos pueden provocar errores o inconsistencia.
 - Se deben implementar mecanismos para evitar esto.

Basic Thread Synchronization

- La solución a estos problemas viene con el concepto de SECCION CRITICA (SC).
 - Una SC es un bloque de código que accede a un recurso y NO puede ser ejecutada por mas de un hilo al mismo tiempo.
 - Para facilitar la implementación de una SC, java ofrece mecanismos de sincronización.
 - Cuando un hilo quiere acceder a una SC utiliza uno de los mecanismos de sincronización provistos.

Basic Thread Synchronization

- Cuando más de un hilo estan esperando que un hilo finalice la ejecución de la SC, la JVM selecciona uno y el resto espera su turno.
- Veremos el concepto de la palabra reservada

SYNCHRONIZED

Synchronizing a Method

- En este Ej. aprenderemos a utilizar uno de los métodos básicos de sincronización.
- El uso de la palabra clave `synchronized` para controlar el acceso concurrente a un método.
- **Solo un hilo en ejecución puede acceder a un método que tenga en su definición la palabra `synchronized`.**

Synchronizing a Method

- Los métodos estáticos tienen otro comportamiento. Solo un hilo puede acceder un método estático `synchronized`; pero otro hilo puede acceder un método no estático `synchronized`. Si ambos métodos modifican mismas variables puede existir inconsistencia.
- Se verá un ejemplo.
 - 2 Hilos – Uno transfiere y el otro retira de una cuenta.
 - 1 cuenta bancaria.
 - Garantizar balance final de la cuenta.

Synchronizing a Method

- Paso a paso:
 - Crear la clase Account que modelará la cuenta. Tiene solo un atributo tipo double llamado balance.
 - Implementar los métodos SetBalance y GetBalance para escribir y leer el valor de la cuenta.
 - Implementar el método addAmount() que agregará dinero a la cuenta en el importe enviado al método.
 - Solo un hilo puede cambiar el valor de la cuenta a la vez. Por lo que deberá utilizar la palabra synchronized en este método.
 - Implementar el método subtractAmount(). Idem

Synchronizing a Method

- Implementar la clase que simula una TAS. Esta clase debe implementar runnable.
 - Agregar un objeto de la clase account.
 - Implementar el método run(). El mismo hará 100 llamados al método subtract value.
- Implementar la clase que simula una compañía que transfiere dinero.
 - Agregar un objeto de la clase account.
 - Implementar el método run(). El mismo hará 100 llamados al método addAmount().

Synchronizing a Method

- Crear la clase Main.
 - Crear un objeto de la clase account.
 - Inicializarlo en un valor.
 - Crear un objeto Compania y un Thread para ejecutarlo.
 - Crear un objeto Banco y un Thread para ejecutarlo.
- Imprimir el mensaje inicial en la consola.
- Esperar la finalización de los Threads (join).
- Imprimir el valor final en la consola.

Synchronizing a Method

- Como funciona:
- En el ejercicio se incrementa y decrementa el valor de una cuenta en igual cantidad de veces y valor.
- Es esperable que el valor final de la cuenta sea igual al inicial.
- Hemos intentado forzar un error usando una variable temporal e introduciendo un delay.
- Si quiere observar los problemas de la concurrencia elimine las palabras synchronized de los métodos que modifican la cuenta.

Synchronizing a Method

- La palabra clave synchronized penaliza la performance. Debe utilizarse solo en los casos necesarios.
- La palabra synchronized puede utilizarse para proteger un bloque de código y NO todo el método.
- SC as short as possible.

Synchronizing a Method

- Para estos casos, se debe enviar un objeto como llave para el synchronize del bloque de código.
- Generalmente, se usa la palabra reservada THIS.

```
synchronized (this) {  
    // Java code  
}
```

Synchronizing a Method

- El código:

```

Main.java Account.java Bank.java Company.java
1 package com.packtpub.java7.concurrency.chapter2.recipe1.core;
2
3 import com.packtpub.java7.concurrency.chapter2.recipe1.task.Account;
4
5
6
7
8 * Main class of the example. It creates an account, a company and a bank
9
10 public class Main {
11
12
13
14 * Main method of the example
15
16 public static void main(String[] args) {
17
18 // Creates a new account ...
19 Account account=new Account();
20 // an initialize its balance to 1000
21 account.setBalance(1000);
22
23
24 // Creates a new Company and a Thread to run its task
25 Company company=new Company(account);
26 Thread companyThread=new Thread(company);
27 // Creates a new Bank and a Thread to run its task
28 Bank bank=new Bank(account);
29 Thread bankThread=new Thread(bank);
30
31 // Prints the initial balance
32 System.out.printf("Account : Initial Balance: %f\n",account.getBalance());
33
34 // Starts the Threads
35 companyThread.start();
36 bankThread.start();
37
38 try {
39 // Wait for the finalization of the Threads
40 companyThread.join();
41 bankThread.join();
42 // Print the final balance
43 System.out.printf("Account : Final Balance: %f\n",account.getBalance());
44 } catch (InterruptedException e) {
45 e.printStackTrace();
46 }
47
48 }
```

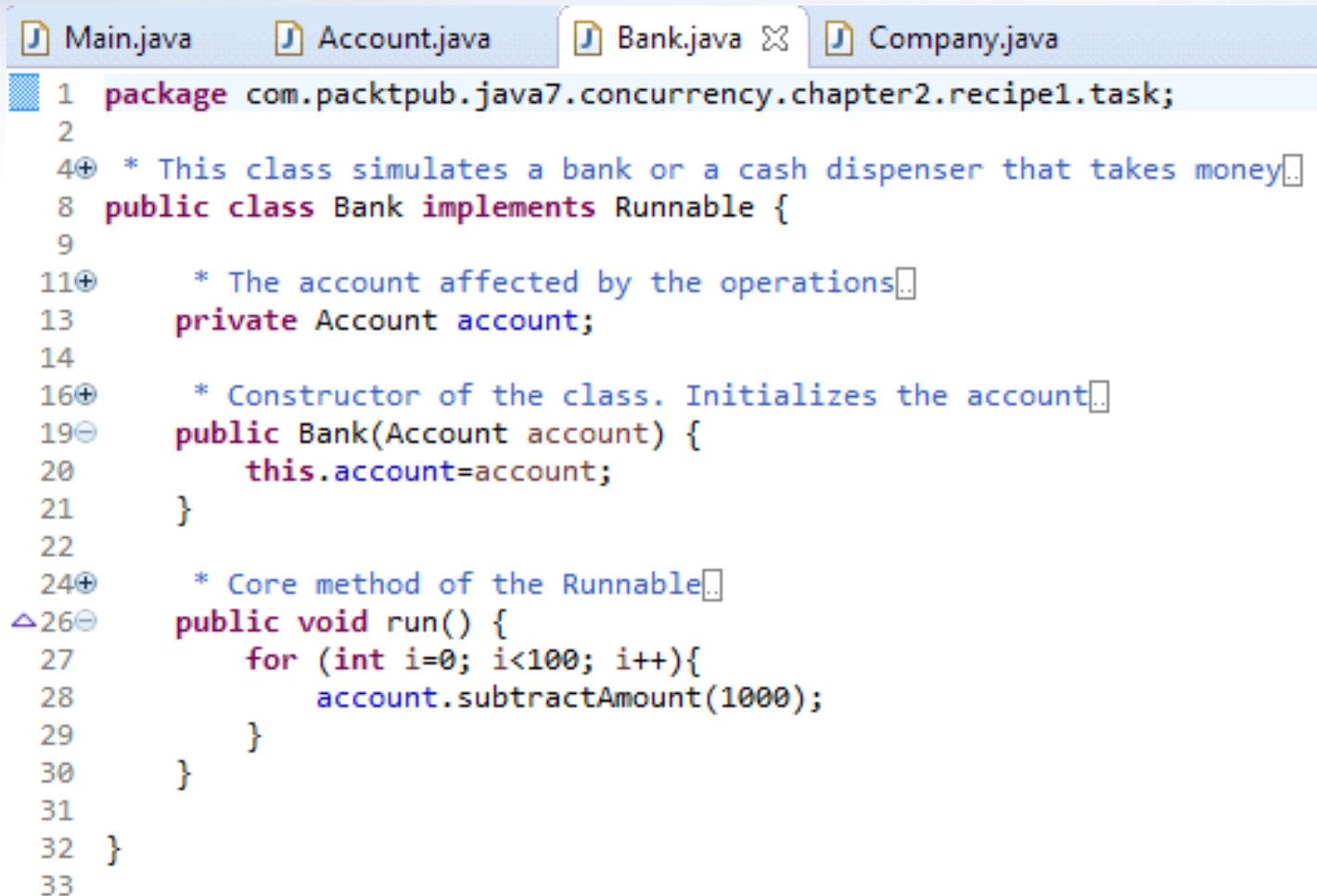
Synchronizing a Method

- El código:

```
Main.java Account.java Bank.java Company.java
1 package com.packtpub.java7.concurrency.chapter2.recipe1.task;
2
3
4 * This class simulates a bank account
5
6 public class Account {
7
8
9     * Balance of the bank account
10     private double balance;
11
12
13     * Returns the balance of the account
14     public double getBalance() {
15         return balance;
16     }
17
18
19     * Establish the balance of the account
20     public void setBalance(double balance) {
21         this.balance = balance;
22     }
23
24
25     * Add an import to the balance of the account
26     public synchronized void addAmount(double amount) {
27         double tmp=balance;
28         try {
29             Thread.sleep(10);
30         } catch (InterruptedException e) {
31             e.printStackTrace();
32         }
33         tmp+=amount;
34         balance=tmp;
35     }
36
37
38     * Subtract an import to the balance of the account
39     public synchronized void subtractAmount(double amount) {
40         double tmp=balance;
41         try {
42             Thread.sleep(10);
43         } catch (InterruptedException e) {
44             e.printStackTrace();
45         }
46         tmp-=amount;
47         balance=tmp;
48     }
49 }
```

Synchronizing a Method

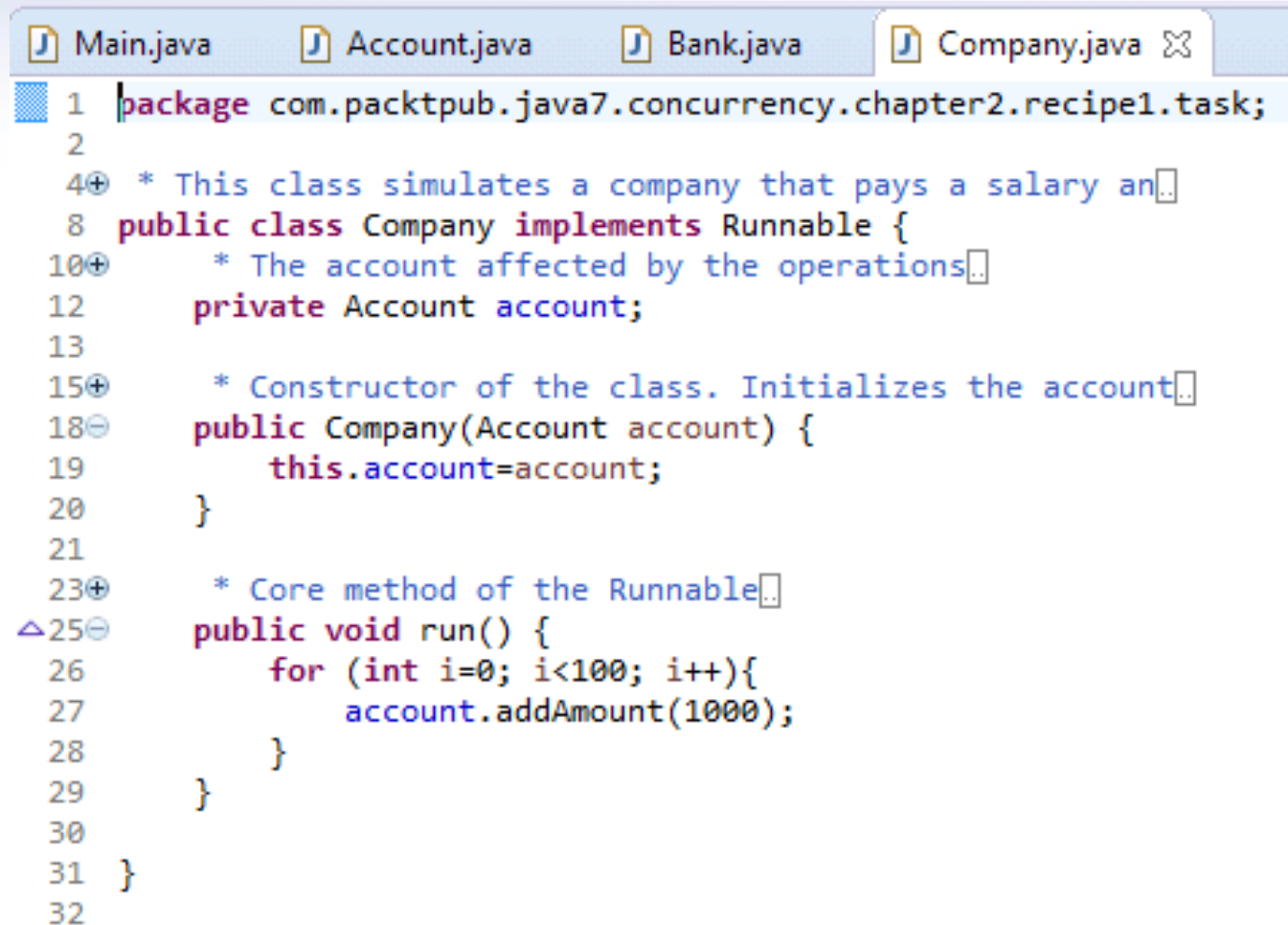
- El código:



```
1 package com.packtpub.java7.concurrency.chapter2.recipe1.task;
2
3
4+ * This class simulates a bank or a cash dispenser that takes money.
5
6
7
8 public class Bank implements Runnable {
9
10
11+ * The account affected by the operations.
12
13 private Account account;
14
15
16+ * Constructor of the class. Initializes the account.
17
18
19- public Bank(Account account) {
20     this.account=account;
21 }
22
23
24+ * Core method of the Runnable.
25
26- public void run() {
27     for (int i=0; i<100; i++){
28         account.subtractAmount(1000);
29     }
30 }
31
32 }
33
```

Synchronizing a Method

- El código:



```
1 package com.packtpub.java7.concurrency.chapter2.recipe1.task;
2
3
4 * This class simulates a company that pays a salary an
5
6
7 public class Company implements Runnable {
8
9     * The account affected by the operations
10
11     private Account account;
12
13
14     * Constructor of the class. Initializes the account
15
16     public Company(Account account) {
17         this.account=account;
18     }
19
20
21
22     * Core method of the Runnable
23
24     public void run() {
25         for (int i=0; i<100; i++){
26             account.addAmount(1000);
27         }
28     }
29 }
30
31
32
```


Synchronizing a Method


- La salida con y sin **synchronized**:

 Console 

<terminated> Main (36) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (17 de abr. de 2017 8:37:01 p. m.)

Account : Initial Balance: 1000.000000

Account : Final Balance: 1000.000000

 Console 

<terminated> Main (36) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (17 de abr. de 2017 8:37:42 p. m.)

Account : Initial Balance: 1000.000000

Account : Final Balance: -85000.000000

Arranging independent attributes in synchronized classes

- Cuando se utiliza la palabra `synchronized` para proteger un bloque de código, debe pasar un objeto referencia como parámetro.
- Normalmente se usa el argumento `THIS`, para referenciar el objeto que ejecuta ese método.
 - Puede usarse otro objeto como referencia.
- Por ej. si tiene dos atributos independientes en una clase compartida por múltiples hilos debe sincronizar el acceso a ellas.

Arranging independent attributes in synchronized classes

- En el próximo ejercicio resolvera una situación de simulación de un cine con dos pantallas y dos oficinas de venta.
 - Cuando una oficina vende una entrada es para una pantalla no para ambas.
 - La cantidad de asientos libres de cada sala es INDEPENDIENTE.
 - Paso a paso

Arranging independent attributes in synchronized classes

- Crear la clase cinema. Agregar 2 atributos que representarán los asientos vacíos de ambas salas.
 - Agregar a la clase cinema 2 objetos adicionales que se usarán para el control del código SC.
 - Implementar el constructor.
 - Implementar el método sellTickets1. El mismo es llamado cuando se vende una entrada de esta sala. Y se utiliza el objeto controlCinema1 para SC.
 - Implementar el método sellTickets2. Idem con controlCinema2.

Arranging independent attributes in synchronized classes

- Implementar el método returnTickets1. Método que se llama cuando se devuelven entradas de la sala 1.
- Implementar el método returnTickets2. Idem.
- Implementar dos métodos que devolverán el número de vacantes de ambas salas.
- Implementar la clase TicketOffice1, runnable.
 - Declarar un objeto cinema, y el constructor.
 - Implementar el método run(). Y simular algunas operaciones con ambas salas.
- Implementar idem TicketOffice2.

Arranging independent attributes in synchronized classes

- Implementar la clase main.
 - Declarar y crear un objeto cinema.
 - Crear un objeto de la clase TicketOffice1 y un Thread.
 - Idem TicketOffice2.
 - Lanzar ambos hilos.
 - Esperar la finalización de los hilos con join.
 - Escribir en la consola las vacantes de ambas salas.

Arranging independent attributes in synchronized classes

- Como funciona:
 - Cuando se utiliza la palabra synchronized para proteger un bloque de código se usa un objeto como parametro.
 - JVM garantiza que solo un hilo puede acceder a todos los bloques de código protegidos con ese objeto.
 - El codigo:

Arranging independent attributes in synchronized classes

- Clase

TicketOffice1

```

Main.java Cinema.java TicketOffice1.java TicketOffice2.java
1 package com.packtpub.java7.concurrency.chapter2.recipe2.task;
2
3
4 * This class simulates a ticket office. It sell or return tickets.
5
6
7 public class TicketOffice1 implements Runnable {
8
9
10
11 * The cinema
12 private Cinema cinema;
13
14
15
16 * Constructor of the class
17 public TicketOffice1 (Cinema cinema) {
18     this.cinema=cinema;
19 }
20
21
22
23
24 * Core method of this ticket office. Simulates selling and returning tickets
25
26 @Override
27 public void run() {
28     cinema.sellTickets1(3);
29     cinema.sellTickets1(2);
30     cinema.sellTickets2(2);
31     cinema.returnTickets1(3);
32     cinema.sellTickets1(5);
33     cinema.sellTickets2(2);
34     cinema.sellTickets2(2);
35     cinema.sellTickets2(2);
36 }
37
```


Arranging independent attributes in synchronized classes

- Class

TicketOffice2

```

Main.java Cinema.java TicketOffice1.java TicketOffice2.java ✕
1 package com.packtpub.java7.concurrency.chapter2.recipe2.task;
2
4+ * This class simulates a ticket office. It sell or return tickets.
8 public class TicketOffice2 implements Runnable {
9
11+ * The cinema .
13 private Cinema cinema;
14
16+ * Constructor of the class.
19- public TicketOffice2(Cinema cinema){
20     this.cinema=cinema;
21 }
22
24+ * Core method of this ticket office. Simulates selling and returning tickets.
26- @Override
27 public void run() {
28     cinema.sellTickets2(2);
29     cinema.sellTickets2(4);
30     cinema.sellTickets1(2);
31     cinema.sellTickets1(1);
32     cinema.returnTickets2(2);
33     cinema.sellTickets1(3);
34     cinema.sellTickets2(2);
35     cinema.sellTickets1(2);
36 }
37
```

Arranging independent attributes in synchronized classes

- Clase Cinema

```

Main.java Cinema.java TicketOffice1.java TicketOffice2.java
3 public class Cinema {
4
5     * This two variables store the vacancies in two cinemas.
6     private long vacanciesCinema1;
7     private long vacanciesCinema2;
8
9     * Two objects for the synchronization. ControlCinema1 synchronizes the
10    private final Object controlCinema1, controlCinema2;
11
12    * Constructor of the class. Initializes the objects.
13    public Cinema(){
14        controlCinema1=new Object();
15        controlCinema2=new Object();
16        vacanciesCinema1=20;
17        vacanciesCinema2=20;
18    }
19
20    * This method implements the operation of sell tickets for the cinema 1.
21    public boolean sellTickets1 (int number) {
22        synchronized (controlCinema1) {
23            if (number<vacanciesCinema1) {
24                vacanciesCinema1-=number;
25                return true;
26            } else {
27                return false;
28            }
29        }
30    }
31
32    * This method implements the operation of sell tickets for the cinema 2.
33    public boolean sellTickets2 (int number){
34        synchronized (controlCinema2) {
35            if (number<vacanciesCinema2) {
36                vacanciesCinema2-=number;
37                return true;
38            } else {
39                return false;
40            }
41        }
42    }
43
44    }
45
46    }
```

Arranging independent attributes in synchronized classes

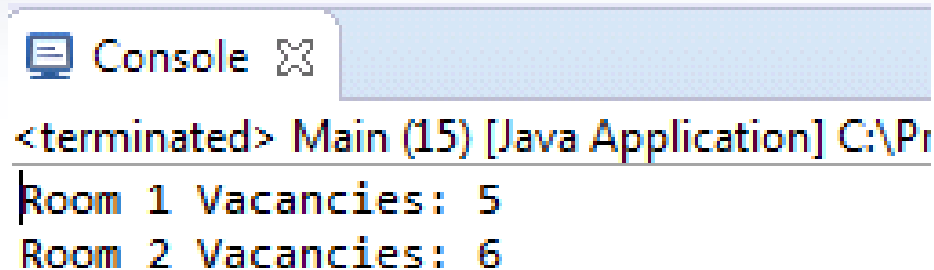
- Clase Main

```
Main.java  Cinema.java  TicketOffice1.java  TicketOffice2.java

1  package com.packtpub.java7.concurrency.chapter2.recipe2.core;
2
3  import com.packtpub.java7.concurrency.chapter2.recipe2.task.Cinema;
4
5
6
7  * Core class of the example. Creates a cinema and two threads for
8
9
10
11
12  public class Main {
13
14
15      * Main method of the example
16
17      public static void main(String[] args) {
18          // Creates a Cinema
19          Cinema cinema=new Cinema();
20
21
22          // Creates a TicketOffice1 and a Thread to run it
23          TicketOffice1 ticketOffice1=new TicketOffice1(cinema);
24          Thread thread1=new Thread(ticketOffice1,"TicketOffice1");
25
26          // Creates a TicketOffice2 and a Thread to run it
27          TicketOffice2 ticketOffice2=new TicketOffice2(cinema);
28          Thread thread2=new Thread(ticketOffice2,"TicketOffice2");
29
30          // Starts the threads
31          thread1.start();
32          thread2.start();
33
34          try {
35              // Waits for the finalization of the threads
36              thread1.join();
37              thread2.join();
38          } catch (InterruptedException e) {
39              e.printStackTrace();
40          }
41
42          // Print the vacancies in the cinemas
43          System.out.printf("Room 1 Vacancies: %d\n",cinema.getVacanciesCinema1());
44          System.out.printf("Room 2 Vacancies: %d\n",cinema.getVacanciesCinema2());
45      }
46
47  }
```

Arranging independent attributes in synchronized classes

- Resultados Esperados



The screenshot shows a console window titled "Console" with a close button. The output text is as follows:

```
<terminated> Main (15) [Java Application] C:\Pr  
Room 1 Vacancies: 5  
Room 2 Vacancies: 6
```