Laboratorio A.E.D. Laboratorio 1

Guillermo Román
guillermo.roman@upm.es

Julio García
juliomanuel.garcia@upm.es

Lars-Åke Fredlund
larsake.fredlund@upm.es

Tonghong Li
tonghong.li@upm.es

Manuel Carro
manuel.carro@upm.es

Marina Álvarez
marina.alvarez@upm.es

Normas

- Fechas de entrega y penalización:

Hasta el Martes 27 de Septiembre, 12:00 horas	0 %
Hasta el Miércoles 28 de Septiembre, 12:00 horas	20 %
Hasta el Jueves 29 de Septiembre, 12:00 horas	40 %
Hasta el Viernes 30 de Octubre, 12:00 horas	60 %

  Después la puntuación máxima será 0
- Se comprobará plagio y se actuará sobre los detectados.
- Usad las horas de tutoría para preguntar sobre programación – son oportunidades excelentes para aprender.

Entrega

- Todos los ejercicios de laboratorio se deben entregar a través de

`http://deliverit.fi.upm.es`

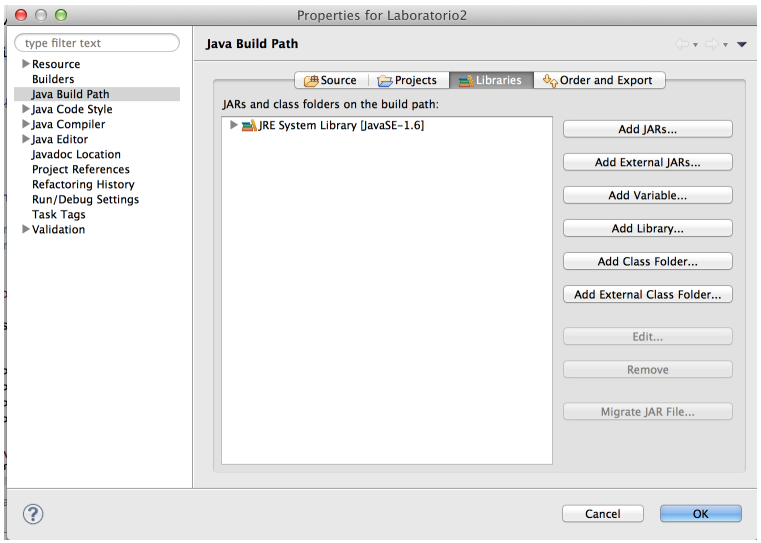
- **Nota: La dirección ha cambiado con respecto con al año pasado.**
- El fichero que hay que subir es `MiHotel.java`.

Configuración previa

- Arrancad Eclipse
- Si trabajáis en portátil, podéis utilizar cualquier versión reciente de Eclipse. Es suficiente con que instaléis la *Eclipse IDE for Java Developers*.
- Cambiad a "Java Perspective".
- Debéis tener instalado al menos Java JDK 8.
- Cread un proyecto Java llamado `aed`:
  - Seleccionad separación de directorios de fuentes y binarios.
  - **No debéis elegir la opción de crear el fichero** `module-info.java`
- Cread un *package* `aed.hotel` en el proyecto `aed`, dentro de `src`
- Aula Virtual → AED → Laboratorios → Laboratorio 1 → `Laboratorio1.zip`; descomprimidlo
- Contenido de Laboratorio1.zip:
  - `Reserva.java`, `Hotel.java`, `Habitacion.java`, `TesterLab1.java`

Configuración previa

- Importad al paquete `aed.hotel` los fuentes que habéis descargado (`Reserva.java`, `Hotel.java`, `Habitacion.java`, `TesterLab1.java`)
- Añadid al proyecto `aed` la librería `aedlib.jar` que tenéis en Moodle (en Laboratorios).

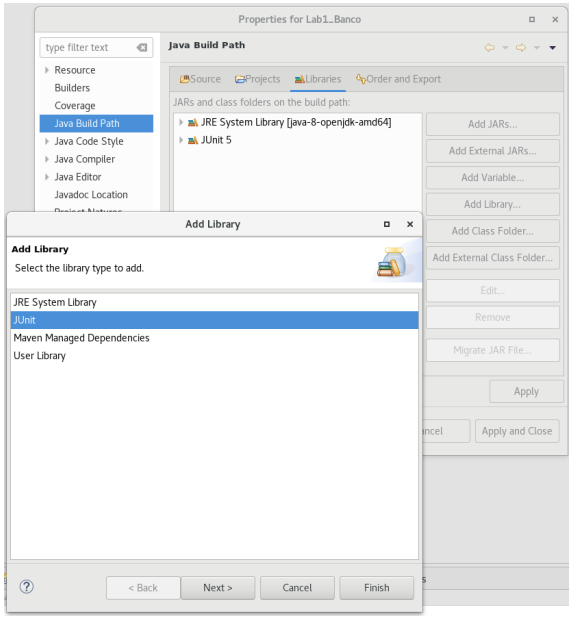


Para ello:

- Project → Properties → Java Build Path. Se abrirá una ventana como la de la izquierda
- Usad la opción “Add External JARs...”.
- Si vuestra instalación distingue `ModulePath` y `ClassPath`, instalad en `ClassPath`

Configuración previa

- Añadid al proyecto `aed` la librería `JUnit 5`



Para ello:

- Project → Properties → Java Build Path. Se abrirá una ventana como la de la izquierda;
- Usad la opción "Add Library..." → Seleccionad "Junit" → Seleccionad "JUnit 5"
- Si vuestra instalacion distingue `ModulePath` y `ClassPath`, instalad en `ClassPath`

- En la clase `TesterLab1` tenéis las pruebas, para ejecutarlas, abrid el fichero `TesterLab1`, pulsando el botón derecho sobre el editor, seleccionar "Run as..." → "JUnit Test"
- NOTA: Si al ejecutar, no aparece la vista "JUnit", podéis incluirla en "Window" → "Show View" → "Java" → "JUnit"

Documentación de la librería aedlib.jar

- La documentación de la API de `aedlib.jar` esta disponible en `http://costa.ls.fi.upm.es/teaching/aed/docs/aedlib/`
- También se puede añadir la documentación de la librería a Eclipse (*no es obligatorio*):
  - En el "Package Explorer": "Referenced Libraries" → aedlib.jar y elige la opción "Properties". Se abre una ventana donde se puede elegir "Javadoc Location" y ahí se pone como "javadoc location path:"

  `http://costa.ls.fi.upm.es/teaching/aed/docs/aedlib/`

  y presionar el buton "Apply and Close"

Tarea: Implementar la interfaz `Hotel`

- Se quiere desarrollar un sistema para gestionar las reservas de habitaciones en un hotel
- Para ello se ha definido la interfaz `Hotel` que dispone de métodos para:
  - Buscar una habitación libre por fechas
  - Hacer una reserva
  - Cancelar una reserva
  - Gestionar las habitaciones del hotel
  - Planificar tareas del hotel como la limpieza de habitaciones

Tarea: Implementar la interfaz Hotel

- Se pide implementar la interfaz Hotel:

```
interface Hotel {
  void anadirHabitacion(Habitacion habitacion);
  IndexedList<Habitacion>
        disponibilidadHabitaciones(String diaEntrada,
                                   String diaSalida);
  boolean reservaHabitacion(Reserva reserva);
  boolean cancelarReserva(Reserva reserva);
  Reserva reservaDeHabitacion(String nombreHabitacion, String dia);

  IndexedList<Reserva> reservasPorCliente(String dniPasaporte);
  IndexedList<Habitacion> habitacionesParaLimpiar(String hoyDia);
}
```

- Documentación detallada en Hotel.java y también en https://costa.ls.fi.upm.es/teaching/aed/docs/practicas/hotel/

La clase `MiHotel`

- La clase debe implementar la interfaz `Hotel`:

```
public class MiHotel implements Hotel {
   ...
}
```

- Debéis usar el atributo `habitaciones` de tipo `IndexedList<Habitacion>` que ya está en el esqueleto de la clase.
- Podéis usar todos los métodos privados que consideréis necesarios
- La clase dispone de un único constructor sin parámetros para inicializar el atributo `habitaciones`, que debe almacenar las habitaciones del hotel
- Revisad los consejos antes de tomar decisiones

La clase Habitacion

`Habitacion` almacena la información de una habitación del hotel:

```
public class Habitacion implements Comparable<Habitacion> {
  private String nombre;                    // un nombre unico
  private int precio;                       // precio al noche
  private IndexedList<Reserva> reservas;    // reservas
  ... // getters, equals, toString
```

- Una habitación tiene asociado un nombre, un precio para cada noche y una lista de reservas
- La clase implementa la interfaz `Comparable<Habitacion>`. El método `compareTo` debe usar el orden lexicográfico de los nombres de las habitaciones para compararlas. Por ejemplo, la habitación con nombre `"102"` $<$ la habitación con nombre `"305"`.

La clase Reserva

- Almacena los datos de una reserva de una habitación:

```
public class Reserva implements Comparable<Reserva> {
  private String habitacion;     // habitacion reservada
  private String dniPasaporte;   // dni o pasaporte del cliente
  private String diaEntrada;     // Dia entrada
  private String diaSalida;      // Dia salida
}
```

- Cada reserva se refiere a una habitación, realizada por una persona identificada con DNI / pasaporte y tiene una fecha de entrada y otra de salida.
- Las fechas están representadas como instancias de `String` en la forma "yyyy-mm-dd" (por ejemplo "2022-09-20").
- Dos fechas `fecha1` y `fecha2` son comparables usando el método `compareTo`. Por ejemplo: `fecha1.compareTo(fecha2)` devuelve un entero < 0 si `fecha1` es anterior a `fecha2`.

Ejemplo

```
Hotel h = new MiHotel();

Habitacion h101 = new Habitacion("101",75); h.anadirHabitacion(h101);
Habitacion h102 = new Habitacion("102",100); h.anadirHabitacion(h102);

h.disponibilidadHabitaciones("2022-09-23","2022-09-25"); // => [h101, h102]

Reserva r1 = new Reserva("101","023893Y", "2022-09-24", "2022-09-26");
h.reservaHabitacion(r1); // => true

h.disponibilidadHabitaciones("2022-09-23","2022-09-25"); // => [h102];
h.disponibilidadHabitaciones("2022-09-23","2022-09-24"); // => [h101,h102]
h.cancelarReserva(r1); // => true
h.disponibilidadHabitaciones("2022-09-23","2022-09-25"); // => [h101,h102]

h.reservaHabitacion(r1); // => true

Reserva r2 = new Reserva("101","023893Y", "2022-09-23", "2022-09-25");
h.reservaHabitacion(r2);  // => false (conflicto)
```

Ejemplo (cont.)

```
Reserva r3 = new Reserva("102","023893Y", "2022-10-01", "2022-10-03");
h.reservaHabitacion(r3); // => true

System.out.println(h);
  MiHotel:
    Habitacion("101",75,Reserva("101",023893Y,"2022-09-24","2022-09-26"))
    Habitacion("102",100,Reserva("102",023893Y,"2022-10-01","2022-10-03"))

// Reservas realizados por un cliente
h.reservasPorCliente("023893Y");          // => [r1,r3]

// Devuelve las habitaciones para limpiar hoy
h.habitacionesParaLimpiar("2022-09-24");  // => []
h.habitacionesParaLimpiar("2022-09-25");  // => [h101]
h.habitacionesParaLimpiar("2022-09-26");  // => [h101]
```

Consejos

Seguir estos consejos os permitirá conseguir mejores resultados!

- ¿Cuándo dos reservas r_1 y r_2 de la misma habitación no están "en conflicto"? Cuando la fecha de salida de salida de r_1 es menor o igual que la fecha de entrada de r_2, o vice versa (la fecha de salida de r_2 es menor o igual que la fecha de entrada de r_1).
- Para obtener la máxima puntuación las habitaciones dentro el atributo `habitaciones` deberían estar ordenadas según sus nombres y las operaciones de añadir una habitación nueva, o buscar una habitación deberían ser implementadas mediante una "búsqueda binaria".

Consejos

Seguir estos consejos os permitirá conseguir mejores resultados!

- Múltiples métodos devuelven una lista nueva, ordenada según diferentes criterios (`reservasPorCliente`, ...). Es una buena idea implementar un método genérico

  ```
  static <E> void insertar(E e,IndexedList<E> l,Comparator<E> cmp)
  ```

  que inserta un elemento `e` dentro `l`, en el orden especificado por el comparador `cmp`. El algoritmo para insertar el elemento debería también ser un búsqueda binaria para obtener la máxima puntuación.
- Notad que para poder usar `insertar` tenéis que implementar clases que implementan la interfaz `Comparator<E>`. Dichas clases se puede poner al final del fichero `MiHotel.java` utilizando clases estáticas:

```
class MiHotel ... {
...
static class MiComparador implements Comparator<Reserva> {
  ...
}
}
```

Consejos

Seguir estos consejos os permitirá conseguir mejores resultados!

- Las reservas (de una habitación) deberían ser ordenadas según la fecha de entrada.
- Sin embargo, no resta puntos hacer búsquedas lineales para por ejemplo detectar reservas en conflicto, o implementar otras operaciones sobre reservas.
- No sólo está permitido, sino recomendado, definir y usar métodos auxiliares para reducir la cantidad de código.

Evaluación

Seguir estos consejos os permitirá conseguir mejores resultados!

- Corrección
- Ausencia de código repetido con la misma funcionalidad (podéis usar métodos auxiliares para evitarlo)
- Concisión del código
- Legibilidad, incluida selección de nombres descriptivos para variables y métodos
- El código debe estar correctamente indentado y con comentarios *útiles* cuando lo veáis necesario
- Eficiencia:
  - Se valorará la complejidad computacional del código
  - Se valorará no iterar innecesariamente en los recorridos de las estructuras de datos

Notas

- El proyecto debe compilar sin errores y debe cumplirse la especificación de los métodos a completar
- Debe pasar todos los test `TesterLab1` correctamente sin mensajes de error
- **Nota:** una ejecución sin mensajes de error y que pase todas las pruebas **no** significa que el método sea correcto (es decir, que funcione bien para cada posible entrada)
- Todos los ejercicios se corrigen manualmente antes de dar la nota final