UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Sistemas Distribuidos

Curso 17/18 :: Laboratorio (extraordinario)

Escuela Superior de Informática

calificación

Este examen consta de 7 preguntas con un total de 25 puntos. En las preguntas tipo test sólo una opción es correcta a menos que se indique algo distinto. No está permitido el uso de calculadora. La duración máxima de este examen será de 90 minutos.

Algunas preguntas proponen modificaciones sobre el enunciado original. Cada pregunta es independiente, por lo que dichas modificaciones no son acumulativas.

Utilice únicamente el espacio reservado, con letra clara (unos 12 puntos). Se valora positivamente la capacidad de síntesis y se penalizan las explicaciones superfluas y/o innecesarias.

Apellidos: SOLUCIÓN Nombre: _____ Grupo: _____

1. [1p] Escribe el nombre del compañero junto al que has realizado la práctica DROBOTS. Si la has realizado tú solo escribe «SOLO».

2. [6p] Durante el curso has escrito varios servidores para alojar los distintos objetos que soportan un jugador de DROBOTS. Escribe un esbozo de implementación en pseudocódigo para el servidor de DROBOTS (solo el método `Ice.Application::run`). Máximo 20 líneas. Las líneas de código adicionales penalizan.

   Existen varias soluciones posibles. Se dan aquí unas pautas comunes a cualquier solución aceptable. El código (o pseudocódigo debería incluir los pasos típicos de un servidor Ice:

   - Creación de un adaptador de objetos.
   - Creación de una instancia del sirviente de la interfaz `drobots::Game`.
   - Registrar (anañdir) el sirviente al adaptador para crear el objeto remoto.
   - Activación del adaptador.
   - Espera hasta la finalización del servidor.

   La parte esencial es la instanciación del sirviente ya que se trata del servidor del juego DROBOTS y por tanto es indispensable crear al menos un objeto `Game`.

   El servidor no debería incluir (penaliza la puntuación):

   - Creación de instancias de Robot, RobotController, Player o cualquier otra entidad del juego.
   - Factorías o contenedores sin especificar su uso, que además añade complejidad no solicitada ni justificable sin requisitos arquitecturales.
   - Invocación de métodos de cualquier otro objeto remoto.

3. [6p] De acuerdo a la especificación de DROBOTS, escribe un esbozo de implementación en pseudocódigo para el sirviente de la interfaz `drobots.Game`. Máximo 20 líneas. Las líneas de código adicionales penalizan.

   Existen varias soluciones posibles. Se dan aquí unas pautas comunes a cualquier solución aceptable.

   El sirviente será una clase (por convenio llamada `GameI`) que hereda de la interfaz `drobots::Game`. El único método público de dicha interfaz es `login()`.

   La implementación de `login()` debería incluir tareas tales como:

   - Comprobar si hay una partida en curso.
   - Verificar que el proxy de `Player` es correcto y no está ya registrado.
   - Comprobar que el nick no está siendo utilizado por otro jugador de la partida.
   - Comprobar el número de jugadores registrados hasta el momento.
   - Almacenar el proxy del jugador.
   - Comprobar si la partida puede empezar.
   - Añadir un mecanismo de arranque diferido de la partida para permitir jugadores adicionales.
   - etc.

4. [1p] ¿Qué ventaja supone incluir las factorías de *RobotController* en un container compartido?

   - ☐ **a**) Es un punto único de fallo.
   - ■ **b**) El *Player* puede encontrar todas la factoría pasando únicamente el proxy del container.
   - ☐ **c**) No tiene ninguna ventaja práctica, se hace así para seguir el patrón de diseño.
   - ☐ **d**) El container solo puede contener factorías que creen objetos idénticos.

5. [1p] ¿Qué limitación tiene una factoría que implemente la siguiente interfaz?

```
module drobots {
  interface RobotControllerFactory {
    RobotController* make(Robot* bot, int id);
    DetectorController* makeDetector (int id);
  };
};
```

- [x] **a**) Todos los objetos estarán necesariamente alojados en el mismo nodo.
- [ ] **b**) Solo puede crear *RobotController* de un tipo específico.
- [ ] **c**) Solo se puede desplegar en el mismo nodo que el *Player*.
- [ ] **d**) Puede crear un número limitado de objetos.

6. [5p] En la especificación original los detectores son clientes puros (no tienen interfaz). Suponga que en una modalidad alternativa de DROBOTS, los detectores fueran objetos y es el jugador el que tiene que invocarlos para conseguir la información de los robots detectados.

   (a) Escriba una interfaz *Slice* válida para esta modalidad:

   Las instancias de Detector deben ofrecer una interface para acceder a la información que poseen, que según el enunciado es:
   - El número de robots detectados en el último turno.
   - La posición en la que se encuentra el detector.

   Una posible interfaz (aunque no la única) podría ser:

```
interface Detector {
Point getPosition();
int getDetectedRobots();
};
```

   (b) Proponga una modificación de la interfaz *Player* que permita al jugador obtener las referencias de los detectores:

   La instancia de `Player` requiere un método para que el servidor del juego le comunique las referencias de los objetos Detectores. El método a añadir podría ser similar al siguiente (aunque otras variantes podrían ser aceptables):

```
sequence<Detector*>DetectorSeq;
interface Player {
[...]
void setDetectors(DetectorSeq detectors);
[...]
};
```

   En esta variante los `DetectorController` no tienen sentido y deberían eliminarse de la interfaz, como parte de esta modificación.

7. [5p] Según el enunciado, el servidor del juego pide al *Player* instancias de *DetectorController* e indica: «A diferencia de los controladores para los robots, en este caso un mismo objeto podría recibir los eventos de todos los detectores, por lo que con instanciar un controlador de detector sería suficiente».

   Indique, referenciando su código (ficheros y líneas) qué solución ha empleado. Si ha creado varios detectores, o no los ha implementado es su código, indique qué modificación haría para utilizar crear solo uno.

   Esta pregunta es dependiente del código de cada alumno, por lo que se califica en función de la explicación del código referencia y la solución propuesta en cada caso.