Examen de Programación 3 - Julio 2019 - 2ª parte - Cuestiones

- *Tiempo:* 75 minutos
- Puede haber más de una respuesta correcta. Las respuestas incorrectas no restan, pero hacen que la pregunta puntúe cero puntos.
- Contesta en los espacios previstos al efecto.

Las preguntas siguientes se refieren al diagrama de clases adjunto en la figura 1, que es correcto salvo error u omisión. Representa un modelo parcial de un mundo tridimensional de bloques, especificados por las interfaces `World` y `Block` y sus subclases. Los bloques del mundo son de algún material determinado (tipo enumerado `Material`) y cuando son creados ocupan una posición fija en el mundo, especificada por la clase `Location`. Los bloques pueden ser destruídos (método `Block.breaks()`) y liberar materiales representados por la clase `ItemStack`. En este mundo cohabitan una colección heterogénea de seres vivos (interfaz `LivingEntity`) entre los cuales se encuentran los jugadores (interfaz `Player`), que pueden realizar ciertas acciones en el mundo (no especificadas aquí) y los `Mob`, que tienen la obsesión de perseguir a los jugadores y acabar con ellos.

Más allá de lo que aparezca en los fragmentos de código del enunciado, el cometido e implementación de cada método no es relevante para contestar las preguntas, pero sí lo son las relaciones entre las diferentes entidades que aparecen en el diagrama. Asume que los métodos que no están implementados, en caso de poder ser ejecutados, no lanzarán excepciones ni emitirán nada por la salida estándar. Estudia bien el diagrama antes de contestar las preguntas. Todas las clases están en el mismo paquete, incluída la clase `Main` que aparece en algunas preguntas.

Extracto de código de la jerarquía de `Block`.

```
interface Block {
  boolean breaks();
  Collection<ItemStack> getDrops();
  boolean isPassable();
  Material getType();
  Location getLocation();
}

public abstract class AbstractBlock implements Block {
  private Material type;
  private Location loc;
...
  public boolean isPassable() { return false; }
  public String toString() {
    return "Block["+type+":"+loc.getX()+","+loc.getY()+","+loc.getZ()+"]";
    // p. ej. "Block[BEDROCK:0,0,1]"
  }
}

public class RedstoneBlock extends AbstractBlock {
...
 public boolean isPassable() { return true; }
 public boolean isPowered() { return true; }
}
```

1. (0.3 puntos) Al invocar alguno de los constructores de BedRockBlock...
   - ◯ BedRockBlock() puede invocar implícitamente a AbstractBlock()
   - ◯ BedRockBlock(Location) invocará explícitamente a AbstractBlock(Material, Location)
   - ◯ BedRockBlock() puede invocar implícitamente a AbstractBlock() y éste hacer lo mismo con Block()
   - ◯ BedRockBlock(Location) invocará explícitamente a AbstractBlock(Material, Location) y éste podrá invocar implícitamente a Block()

2. (0.3 puntos) Cuando ejecutamos '`new RedstoneBlock(new Location(...))`'...
   - ◯ Primero se ejecuta RedstoneBlock(Location) y luego AbstractBlock(Material, Location)
   - ◯ Primero se ejecuta RedstoneBlock(Location) y luego AbstractBlock()
   - ◯ Primero se ejecuta AbstractBlock(Material, Location) y luego RedstoneBlock(Location)
   - ◯ Primero se ejecuta AbstractBlock() y luego RedstoneBlock(Location)
3. (0.3 puntos) Dada esta asignación: '`Block b = new RedstoneBlock(new Location(...));`'
   - ◯ podemos ejecutar '`b.isPassable()`'
   - ◯ podemos ejecutar '`b.isPowered()`'
   - ◯ no podemos ejecutar '`b.isPassable()`'
   - ◯ no podemos ejecutar '`b.isPowered()`'
4. (0.3 puntos) Si tenemos la asignación '`Block b = new RedstoneBlock(new Location(...));`', el resultado de llamar a '`b.isPassable()`' es
   - ◯ el método devuelve `true`
   - ◯ el método devuelve `false`
   - ◯ un error de ejecución
   - ◯ un error de compilación
5. (0.3 puntos) Si tenemos la asignación '`RedstoneBlock b = new RedstoneBlock(new Location(...));`', el resultado de llamar a '`b.isPassable()`' es
   - ◯ el método devuelve `true`
   - ◯ el método devuelve `false`
   - ◯ un error de ejecución
   - ◯ un error de compilación
6. (0.3 puntos) Si tenemos las siguientes instrucciones: '`RedstoneBlock b = (RedstoneBlock) new Block(); b.isPassable();`', el resultado es
   - ◯ La llamada a `isPassable()` devuelve true
   - ◯ La llamada a `isPassable()` devuelve false
   - ◯ un error de ejecución
   - ◯ un error de compilación

7. (2 puntos) Implementa los siguientes constructores:

   - `AbstractBlock(Material, Location)`,
   - `BedRockBlock(Location)`,
   - `AbstractBlock(AbstractBlock)`,
   - `BedRockBlock(BedRockBlock)`

   (hazlo en esta misma página).

En los fragmentos de código de las siguientes preguntas, asumimos que existen las sentencias `import` adecuadas para usar las librerías estándar de Java (como `java.util`) y que pretendemos ejecutar el método '`main`'. Asume que todas las clases implementan un método `toString()` que simplemente devuelve una cadena con el nombre de la clase. Para cada una de las siguientes cuestiones indica una sola de estas tres posibles respuestas.

- "Error de compilación en la línea ___." y a continuación explica brevemente porqué se produce el error.
- "Error de ejecución en la línea ___." y a continuación explica brevemente porqué se produce el error.
- "Ejecuta correctamente" y a continuación indica la salida emitida por el programa tras ejecutar el método `main`.

En caso de que existan errores en distintas líneas del código, indica sólo el primero de ellos.

8. (0.3 puntos) Código:

```
1  public class Main {
2    public static void main(String[] args) {
3        World world = new WorldImpl();
4        List<RedstoneBlock> list = new ArrayList<>();
5        for (int i=0; i<5; i++)
6          list.add(new RedstoneBlock(new Location(world, 0,0,i)));
7            // Guardamos la lista de bloques en el mundo para luego recuperarlos por su
8            // posicion (Location) con el metodo getBlockAt(Location)
9        world.addBlocks(list);
10           System.out.println(world.getBlockAt(new Location(world, 0,0,0)));
11   }
12 }
```

Respuesta:

9. (0.3 puntos) Código:

```
1  public class Main {
2    public static void main(String[] args) {
3           World w = new WorldImpl();
4           List<LivingEntity> lves = w.getLivingEntities();
5           for (LivingEntity e : lves) {
6             if (e instanceof Player)
7               w.addPlayer(e);
8           }
9           System.out.println("Players added to the world.");
10 }
```

Respuesta:

10. (0.3 puntos) Código:

```
1  public class Main {
2    public static void main(String[] args) {
3           World w = new WorldImpl();
4           Player p = null;
5           List<LivingEntity> lves = w.getLivingEntities();
6           for (LivingEntity e : lves) {
7             if (e instanceof Mob)
8               e.setTarget(p);
9           }
10          System.out.println("Mobs have targets now.");
11 }
```

Respuesta:

11. (0.3 puntos) Código:

```
1  public class Main {
2    public static void main(String[] args) {
3           World w = new WorldImpl();
4           List<Location> locs = new ArrayList<>();
5           for (int i=0; i<3; i++)
6               locs.add(new Location(w,i,i,i));
7           List<Block> blocks = new ArrayList<>();
8           blocks.add(new BedRockBlock(locs.get(0)));
9           blocks.add(new RedstoneBlock(locs.get(1)));
10          blocks.add(new RedstoneBlock(locs.get(2)));
11          // Guardamos la lista de bloques en el mundo para luego recuperarlos por su
12          // posicion (Location) con el metodo getBlockAt(Location)
13          w.addBlocks(blocks);
14          for (int i=0; i<3; i++)
15              System.out.println(w.getBlockAt(locs.get(i)).isPassable());
16   }
17 }
```

Respuesta:

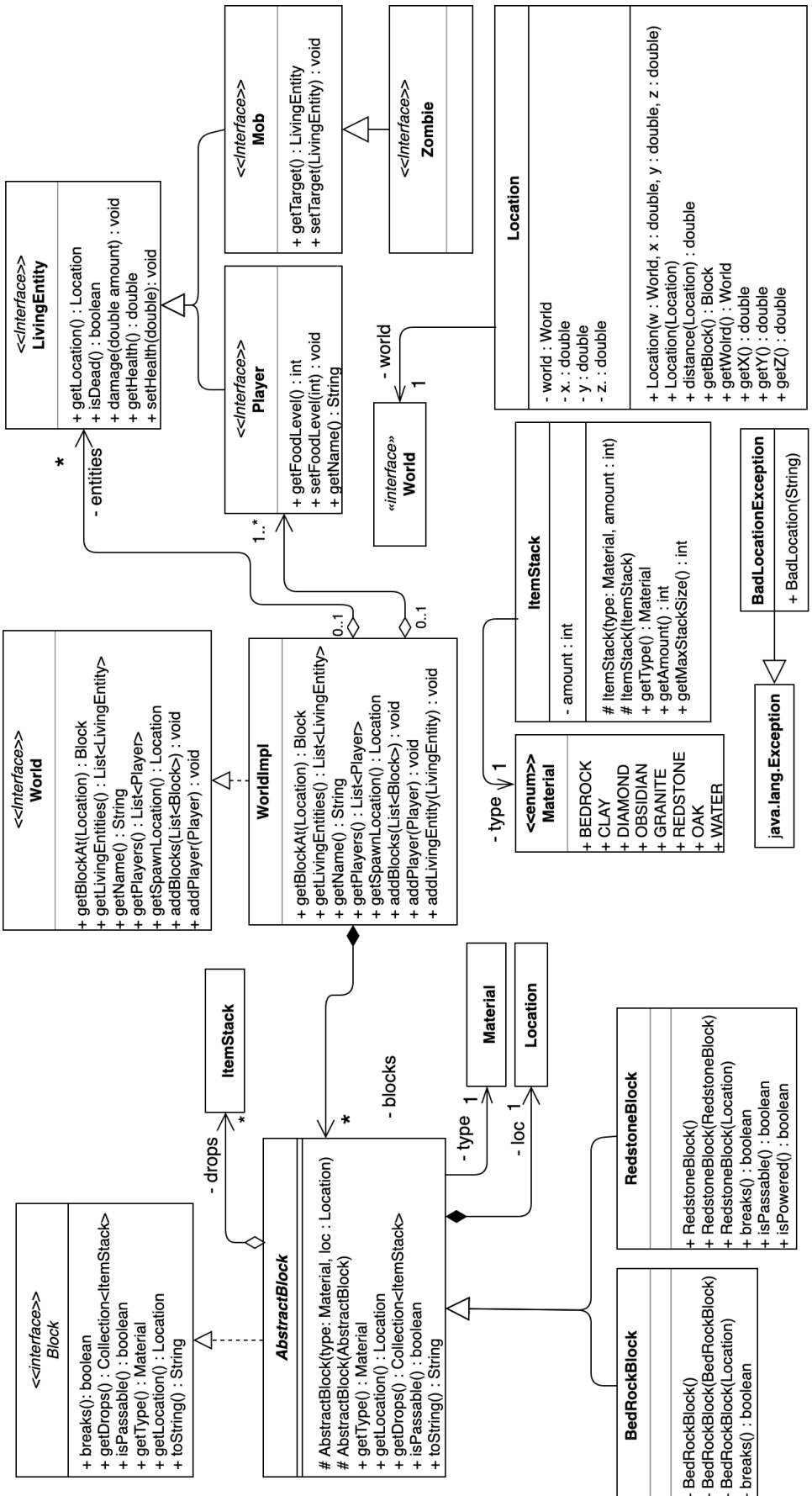

Figura 1: Diagrama de clases.