PROG 3: Práctica 3

PrintStream

File

Scanner

Práctica 3

La clase PrintStream

- Es una de las clases de Java para crear y escribir en ficheros:
  - Constructor:
    - ***PrintStream (String filename)***
- Proporciona utilidades para dar formato a la salida:
    - ***printf (String format, object...args)***
- Tiene métodos *print* y *println* sobrecargados para los tipos primitivos, objetos, cadenas y array de caracteres.

La clase File (I)

- No trabaja sobre un flujo de bytes (datos): trata directamente con el fichero y con el sistema de ficheros
- Proporciona información acerca de los archivos, de sus atributos, de sus directorios, permisos, fechas, etc.
- Tiene 3 constructores:
  - ***File (String name);***
  - ***File (String path, String name);***
  - ***File (File dir, String name);***
- Establecen una conexión con un archivo o directorio físico

La clase File (II)

- Tiene multitud de métodos para extraer información:
  - *canRead, canWrite, exists, isHidden, isDirectory, isFile, createNewFile, getName, list, etc.*
  - *Ejemplo:*

```
File raiz = new File("/usr");
String [] dir = raiz.list();
for (int i=0; dir.length; i++)
    System.out.println(dir[i]);
```

- PrintStream y File pertenecen al paquete ***java.io***

La clase Scanner (I)

- La clase Scanner pertenece al paquete ***java.utils***
- Se introduce a partir de Java 5
- Permite:
  - Definir un patrón delimitador: ***useDelimiter ("-");***
  - Dividir los datos de entrada en subcadenas: ***next()***
  - Convertir esos datos en tipos primitivos como int, float, etc.: ***nextInt(), nextFloat(), ...***
  - Leer hasta un salto de línea: ***nextLine()***

La clase Scanner(II)

- Tres tipos de parámetros para sus constructores:
  - *Scanner (String cadena);*
  - *Scanner (InputStream in);*

    Ejemplo: Scanner(System.in) → Lectura desde entrada estandar
  - ***Scanner (File f); //Lectura desde fichero***

La clase Scanner (III)

- Ejemplo: Supongamos un archivo con esta información:

Kelvin
30
78,7
Horario: 8am a 4pm

Los datos de este archivo pueden ser tratados mediante el siguiente programa:

La clase Scanner(IV)

```
public static void main(String[] args) {
    try {
        Scanner s = new Scanner(new File("archivo.txt"));
        System.out.println(s.nextLine()); //Lee 'Kelvin'
        System.out.println(s.nextInt()); //Lee 30
        System.out.println(s.nextDouble()); //Lee 78,7
        //Lee 'Horario: 8am a 4pm'
        System.out.println(s.nextLine());
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (InputMismatchException e){
        e.printStackTrace();
    }
}
```

Ejemplo sencillo con PrintStream, File, Scanner (I)

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintStream;
import java.util.Scanner;

public class FileHandling {

   public static void main(String[] args) {
      PrintStream ps;
      File f;
      Scanner sc;
      String name= "numbers.txt";
```

Ejemplo sencillo con PrintStream, File, Scanner (II)

```
    try {
      ps= new PrintStream(name);// Vamos a escribir en un fichero
      ps.println("one 1");
      ps.println("two 2");
      ps.close();
      f= new File(name); // Leemos del fichero y escribimos en pantalla
      sc= new Scanner (f);
      System.out.println(sc.next()+ "-" + sc.nextInt());
      System.out.println(sc.next()+ "-" + sc.nextInt());
      sc.close();
    }
    catch (FileNotFoundException e) {
        System.err.println("Error handling files");
    }
  }
}
```

Práctica 3

- **Aparecen**:
  - Nuevos Materiales:
    - Lava y Agua
      - Nuevas constantes del Enum Material.
      - Van al final.
  - Nuevos Bloques:
    - SolidBlock → Recoge la mayor parte del código de Block
    - LiquidBlock → Nuevo bloque formado por los nuevos materiales
  - Nuevas entidades: Animales y Monstruos
    - Heredan de la nueva clase **Creature** que hereda a su vez de **LivingEntity.**

Práctica 3

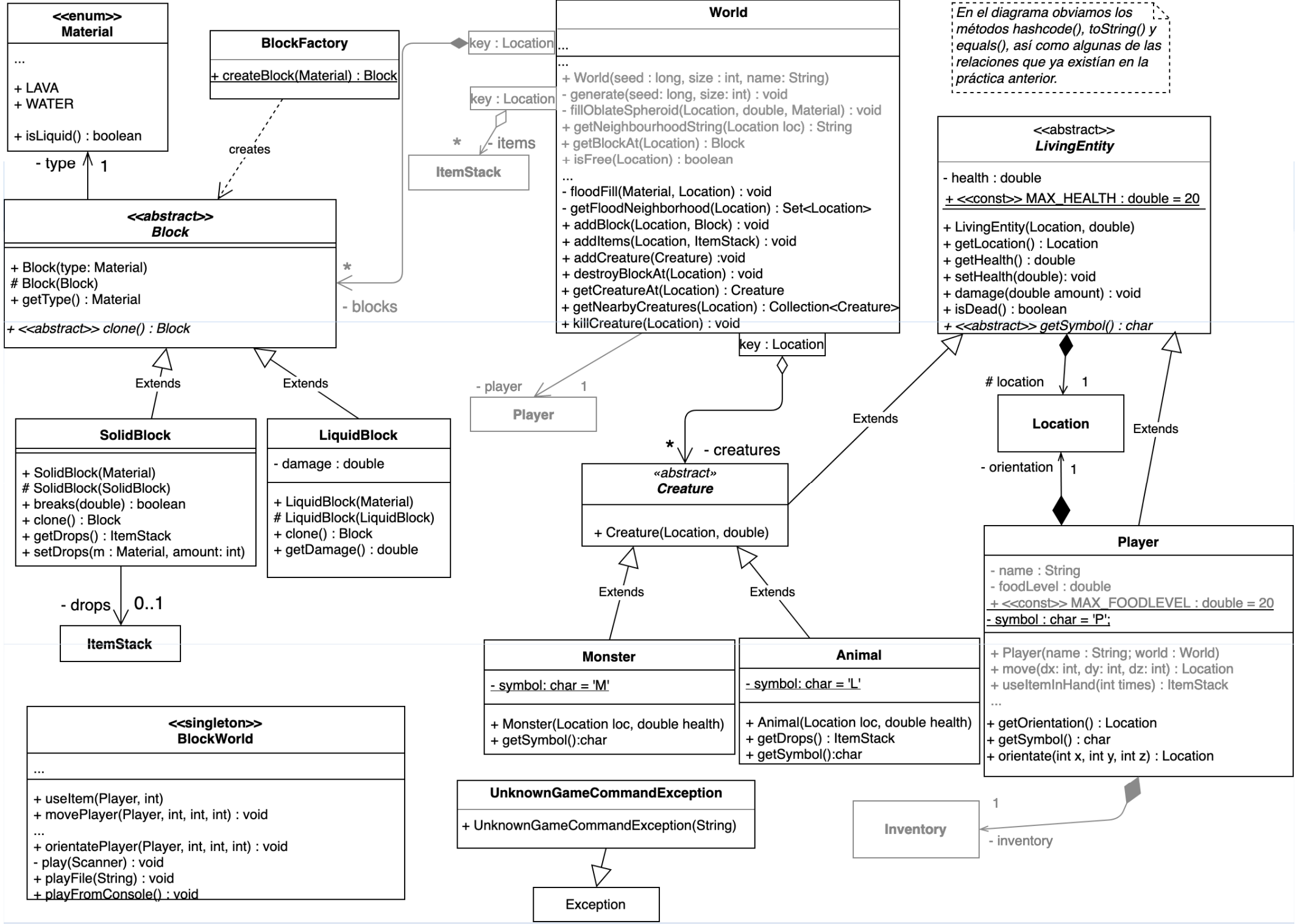
- ***BlockFactory***: Clase para crear los Bloques
- **Cambios**:
  - **Player:**
    - Nuevo atributo: **orientation**
      - Dos posibles implementaciones:
        - **Relativa**: Convertirla a absoluta cuando se necesite.
        - **Absoluta**: Se actualiza en *move(…)* / Se relativiza en *Player.toString();*
    - **useItemInHand**: Actúa sólo sobre la posición adyacente orientada.

Práctica 3

- **Cambios**:
  - **Block**
    - *public abstract Block* ***clone()*** → Copia objeto Block usando su tipo en tiempo de ejecución
  - **World:**
    - Nuevo mapa Map<Location, Creature>
    - Descarga de nuevo:
      - los métodos ***generate()*** y ***fillOblateSpheroid()*** que sustituyen a los de la P2.
      - los nuevos métodos ***floodFill()*** y ***getFloodNeighborhood()*** y los añades a Word

Práctica 3

- **Cambios**:
  - **BlockWorld**
    - ***useItem:***
      - Interacciona según:
        - lo que haya en la posición adyacente orientada y
        - lo que tenga ***player*** en la mano.
      - Ver si debéis modificar **heighMap** al eliminar / añadir un nuevo bloque.
    - Se leera bien desde **fichero** o bien desde **consola**:
      - Los parámetros de generación del mundo.
      - Las instrucciones del juego


«enum»
Material
...
+ LAVA
+ WATER
+ isLiquid() : boolean
BlockFactory
+ createBlock(Material) : Block
creates
- type
1
key : Location
key : Location
* - items
ItemStack
World
...
+ World(seed : long, size : int, name: String)
- generate(seed: long, size: int) : void
- fillOblateSpheroid(Location, double, Material) : void
+ getNeighbourhoodString(Location loc) : String
+ getBlockAt(Location) : Block
+ isFree(Location) : boolean
...
- floodFill(Material, Location) : void
- getFloodNeighborhood(Location) : Set<Location>
+ addBlock(Location, Block) : void
+ addItems(Location, ItemStack) : void
+ addCreature(Creature) :void
+ destroyBlockAt(Location) : void
+ getCreatureAt(Location) : Creature
+ getNearbyCreatures(Location) : Collection<Creature>
+ killCreature(Location) : void
En el diagrama obviamos los métodos hashcode(), toString() y equals(), así como algunas de las relaciones que ya existían en la práctica anterior.
«abstract»
Block
+ Block(type: Material)
# Block(Block)
+ getType() : Material
+ «abstract» clone() : Block
*
- blocks
«abstract»
LivingEntity
- health : double
+ «const» MAX_HEALTH : double = 20
+ LivingEntity(Location, double)
+ getLocation() : Location
+ getHealth() : double
+ setHealth(double): void
+ damage(double amount) : void
+ isDead() : boolean
+ «abstract» getSymbol() : char
Extends
Extends
- player
1
Player
key : Location
location
1
Location
Extends
- orientation
1
* - creatures
«abstract»
Creature
+ Creature(Location, double)
Extends
SolidBlock
+ SolidBlock(Material)
# SolidBlock(SolidBlock)
+ breaks(double) : boolean
+ clone() : Block
+ getDrops() : ItemStack
+ setDrops(m : Material, amount: int)
LiquidBlock
- damage : double
+ LiquidBlock(Material)
# LiquidBlock(LiquidBlock)
+ clone() : Block
+ getDamage() : double
- drops
0..1
ItemStack
Extends
Extends
Player
- name : String
- foodLevel : double
+ «const» MAX_FOODLEVEL : double = 20
- symbol : char = 'P';
+ Player(name : String; world : World)
+ move(dx: int, dy: int, dz: int) : Location
+ useItemInHand(int times) : ItemStack
...
+ getOrientation() : Location
+ getSymbol() : char
+ orientate(int x, int y, int z) : Location
Monster
- symbol: char = 'M'
+ Monster(Location loc, double health)
+ getSymbol():char
Animal
- symbol: char = 'L'
+ Animal(Location loc, double health)
+ getDrops() : ItemStack
+ getSymbol():char
«singleton»
BlockWorld
...
+ useItem(Player, int)
+ movePlayer(Player, int, int, int) : void
...
+ orientatePlayer(Player, int, int, int) : void
- play(Scanner) : void
+ playFile(String) : void
+ playFromConsole() : void
UnknownGameCommandException
+ UnknownGameCommandException(String)
Exception
Inventory
1
- inventory