Módulo	**0789 - Fundamentos da Linguagem Java**
Local	
Sessão	
Formador	
Ficha	`Avaliação`

1. Verdadeiro ou Falso

Construtores são métodos que não possuem um tipo de retorno declarado, devolvendo sempre uma referência.	
Para escrever mensagens para a consola usamos o método System.out.println().	
Uma applet java executa dentro do browser enquanto que uma aplicação java não precisa do browser para executar.	
As applets possuem limitações que permitem manter a segurança dos computadores onde executam.	
As Strings permitem a criação de texto de forma optimizada já que são bastante mutáveis e facilmente alteradas.	
Em Java, os vectores são objectos, mesmo que sejam vectores de tipos primitivos.	
A plataforma Java é composta pela JVM e por uma API com várias bibliotecas de classes.	
A tecnologia Java é independente do hardware, e uma aplicação Java irá funcionar do mesmo modo em qualquer computador ou dispositivo móvel.	
A linguagem de programação Java é uma linguagem orientada a aspectos.	
O JRE é o ambiente necessário à execução das aplicações Java e o JDK necessário à criação de aplicações Java.	
Em linguagem de programação Java existe herança múltipla.	
As interfaces podem agir como marcadores de classes.	
Destrutores servem para manter os objectos em memória.	
Packages são pastas nas quais podemos organizar as classes que construimos.	
O nome completo de uma classe contém o package onde esta se encontra.	
Os modificadores de acesso public, private e protected correspondem, respectivamente, a um acesso público, protegido e privado.	
Se um recurso for marcado como private, nenhuma classe tem acesso a ele, nem mesmo a classe onde o recurso está definido.	
Na programação orientada a objectos os problemas são divididos em objectos que permitem uma aproximação ao mundo real.	
Através da herança de classes é possível criar relações entre objectos que partilham atributos ou comportamentos comuns.	
O processo de encapsulamento permite definir a relação entre os vários objectos, indicando quais são descendentes, quais são apenas usados, etc.	

2. Pacman

Pacman é um dos jogos mais conhecidos e com maior sucesso mundial. É um jogo simples onde o jogador controla um boneco amarelo e percorre os vários labirintos recolhendo as bolinhas e fugindo dos fantasmas que o tentam apanhar. Nesta nova versão do jogo *Pacma* iremos implementar alguns dos níveis e as funcionalidades base que permitem fornecer a experiência de jogo aproximada ao jogo original.

Abra, no IDE, o projecto referente ao jogo a implementar. Este projecto contém já um conjunto de classes implementadas e que ajudarão a construir o jogo. Não se pretende que o jogo seja uma representação fiél do original, e deve focar-se na criação correcta das classes e de alguns métodos base que se encontram em falta (como construtores, getters e setters).

Seguindo o diagrama e as descrições seguintes, comece por implementar as várias classes dentro do *package* **org.sergiolopes.formacao.pacman**.

Figura 1. Diagrama de Classes

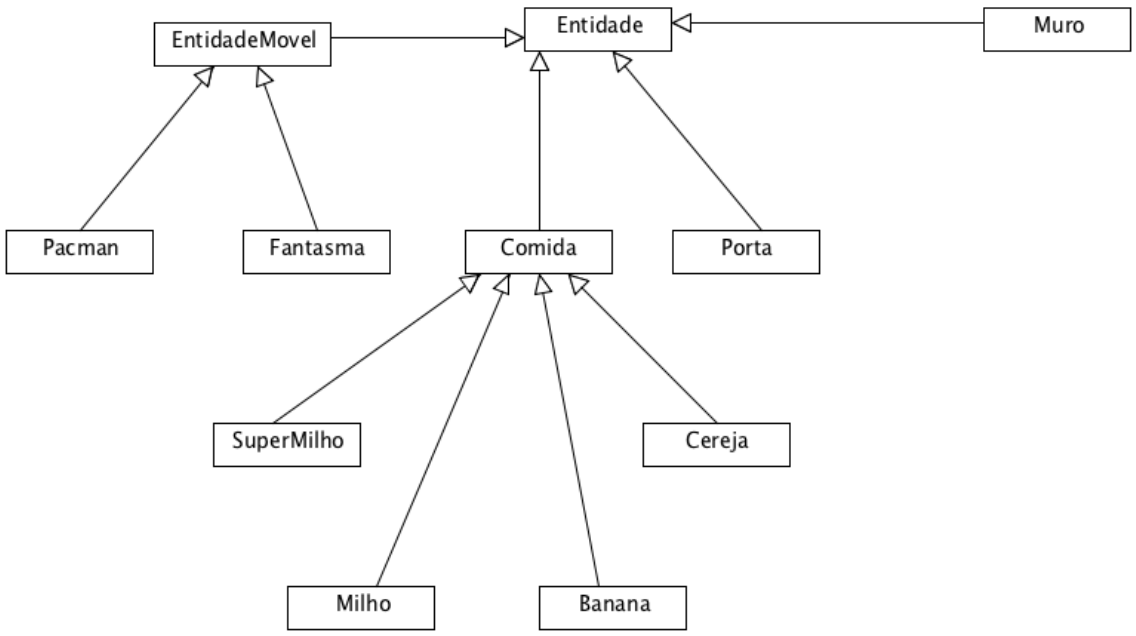

Tabela 1. Descrição das classes a implementar

Banana	Classe que representa uma banana e que pode ser comida pelo pacman.
Cereja	Classe que representa uma cereja e que pode ser comida pelo pacman.
Comida	Classe mãe de todas as classes que representam comida (milho, supermilho, banana e cereja).
Entidade	Classe mãe de todas as classes que possuem uma representação (imagem).
EntidadeMovel	Superclasse de todas as classes que representam elementos móveis no ecrã.
Fantasma	Classe que representa os fantasmas que perseguem o pacman.
Milho	Elemento que oferece alguns pontos ao pacman e que é o objectivo do jogo.
Muro	Representa uma parede que não pode ser atravessada pelo pacman.
Pacman	Classe correspondente ao pacman, esta classe já está implementada e não deve ser criada.
Porta	Classe que representa uma porta que o pacman pode usar para passar.
SuperMilho	Classe que representa um milho maior, com mais pontos que altera os fantamas.

Atributos e Métodos

Classe Entidade:

```
nome, do tipo String
posicao, do tipo Quadricula
animacao, do tipo Animation
jogo, do tipo Jogo
```

```
public void render(GameContainer container, Graphics g) throws SlickException {
    g.drawAnimation(animacao, animacao.getWidth() * posicao.getColuna(),
                animacao.getHeight() * posicao.getLinha());
}
```

Classe Comida:

```
pontos, do tipo int
```

Classe EntidadeMovel:

```
velocidade, do tipo float
x, do tipo float
y, do tipo float
direccao, do tipo Vector2f
```

```
public boolean quadriculaValida(int linha, int coluna) {
    Quadricula possivel = getJogo().getQuadricula(linha, coluna);

    if (possivel == null) {
        return false;
    }

    if (possivel.getMuro() != null) {
        return false;
    }

    if (direccao.getX() == 1 && possivel.getPortaEste() != null) {
        return true;
    }
```

```
    if (direccao.getY() == -1 && possivel.getPortaNorte() != null) {
        return true;
    }

    if (direccao.getX() == -1 && possivel.getPortaOeste() != null) {
        return true;
    }

    if (direccao.getY() == 1 && possivel.getPortaSul() != null) {
        return true;
    }

    return true;

}
```

Classe Fantasma:

```
ESTADO_DORMENTE, static final do tipo int com valor 1
ESTADO_ACTIVO, static final do tipo int com valor 2
ESTADO_FUGIR, static final do tipo int com valor 3
pontos, do tipo int
```

```
public void update(GameContainer container, int delta) throws SlickException {
    float deslocamentoX = ((getVelocidade() * delta * getDireccao().getX()) / 1000f);
    float deslocamentoY = ((getVelocidade() * delta * getDireccao().getY()) / 1000f);

    float x = getX() + deslocamentoX;
    float y = getY() + deslocamentoY;

    int coluna = (int) (x / 27);
    int linha = (int) (y / 27);

    if (quadriculaValida(linha, coluna)) {
        setX(x);
        setY(y);

        Quadricula proxima = getJogo().getQuadricula(linha, coluna);
        getPosicao().removerFantasma(this);

        setPosicao(proxima);
        proxima.adicionarFantasma(this);
    } else {
        switch ((int) (Math.random() * 4)) {
            case 0:
                setDireccao(new Vector2f(1, 0));
                break;
            case 1:
                setDireccao(new Vector2f(-1, 0));
                break;
            case 2:
                setDireccao(new Vector2f(0, 1));
                break;
            case 3:
                setDireccao(new Vector2f(0, -1));
                break;
        }
    }
}
```