การทำงานของโค้ดตรวจสอบไฟล์

```
File file = new File("Example.txt");
if (file.exists()) {
    System.out.println(file.getName() + " exists");
    file.delete();
} else {
    System.out.println("File not found");
}
```

- โปรแกรมจะตรวจสอบว่ามีไฟล์ Example.txt อยู่ใน directory ปัจจุบันหรือไม่
- ถ้ามีไฟล์อยู่ จะแสดงข้อความ "Example.txt exists" จากนั้นจะลบไฟล์
- แต่ถ้าไม่มีไฟล์อยู่ จะแสดงข้อความ “File not found”

การแก้ไขโค้ดให้คอมไพล์ผ่าน

โค้ดจากโจทย์

```
interface Resizable {
    void resize(float scale);
    Resizable() { } // ❌ Error: Interface ไม่สามารถมี Constructor
}

abstract class GraphicObject {
    int xPos, yPos;
    abstract void draw();
}

class Rectangle extends GraphicObject {
    // ❌ ยังไม่ได้ Implement draw()
    @Override
    public void resize(float scale) { /* do something */ }
}

class Circle extends GraphicObject {
    // ❌ ยังไม่ได้ Implement draw()
    @Override
    public void resize(float scale) { /* do something */ }
}
```

โค้ดที่แก้ไขแล้ว

```
interface Resizable {
    void resize(float scale);
}

abstract class GraphicObject {
    int xPos, yPos;
    abstract void draw();
}

class Rectangle extends GraphicObject implements Resizable {
    @Override
    public void resize(float scale) { /* do something */ }
    @Override
    void draw() { /* code */ }
}

class Circle extends GraphicObject implements Resizable {
    @Override
    public void resize(float scale) { /* do something */ }
    @Override
    void draw() { /* code */ }
}
```

สิ่งที่ต้องแก้ไขเพื่อให้โค้ดถูกต้อง

- ลบ constructor ออกจาก interface
- เพิ่ม implements resizable ใน Class Rectangle และ Class Circle เพื่อให้สามารถแก้ไขขนาดได้
- เพิ่ม @Override void draw ทั้งใน Class Rectangle และ Class Circle เพราะเป็น Abstract Method ใน Graphic Object

JavaFXGL Tips L10

Game Application Tasks

- Main Class: Extends GameApplication and overrides key methods:
- initSettings() – ตั้งค่าหน้าต่างของแอปพลิเคชัน
- initGame() – สร้างเอนทิตีของเกม (ผู้เล่น, แผนที่, ศัตรู)
- initInput() – จัดการอินพุตจากผู้ใช้
- initUI() – สร้างอินเทอร์เฟซผู้ใช้ของเกม
- initGameVars() – กำหนดตัวแปรที่ใช้ร่วมกันในเกม
- initPhysics() – ตั้งค่าเครื่องยนต์ฟิสิกส์

Game Resources

- Levels: ไฟล์คำอธิบายแผนที่ (เช่น ไฟล์ .tmx จาก Tiled)
- Textures: กราฟิกของตัวละครและไอเท็ม (เช่น สไปรต์ชีต)
- Sounds: เพลงประกอบและเอฟเฟกต์เสียง
- Assets must be stored in src/main/resources (ทรัพยากรหรือไฟล์ต้องอยู่ใน src/main/resources)

Creating Levels

- ใช้เครื่องมือสร้างแผนที่ Tiled
- ต้องการชุดไทล์ (tileset) สำหรับกราฟิกของแผนที่
- รองรับมุมมองจากด้านบน (top-down) และมุมมองด้านข้าง (side-view)

Map Layers

- ชั้นไทล์ (Tile Layer): วาดแผนที่
- ชั้นวัตถุ (Object Layer): วางวัตถุบนแผนที่
- คุณสมบัติประเภท (Type Property): ใช้สำหรับการสร้างวัตถุใน FXGL
-

การสร้าง Entity Factory:

ต้อง implement EntityFactory เพื่อให้สามารถสร้าง entity โดยอัตโนมัติเมื่อโหลดแผนที่
ใช้ annotation @Spawns("Player") ซึ่งจะต้องตรงกับ "Type" ใน Object Layer ของแผนที่

```
public class MazelaManFactory implements EntityFactory {
    @Spawns("Player")
    public Entity spawnPlayer(SpawnData data) {
        return FXGL.entityBuilder(data)
            .viewWithBBox("player.png")
            .build();
    }
}
```

การโหลดด่าน (Loading the Level)

"Player" จะถูกสร้างโดยอัตโนมัติ
สำหรับวัตถุที่ไม่มี Type ที่ตรงกันในแผนที่ เราต้องสร้าง
มันเองโดยใช้ SpawnData

```
@Override
protected void initGame() {
    FXGL.getGameWorld().addEntityFactory(new MazelaManFactory());
    FXGL.spawn("Background",
        new SpawnData(0, 0).put("width", WIDTH).put("height", HEIGHT));
    FXGL.setLevelFromMap("level1.tmx");
}
```

การจัดการกับ Input จากผู้ใช้
ใช้ FXGL.onKey() สำหรับการจัดการ input พื้นฐาน
ทำงานได้ดีเมื่อ Entity ไม่ได้ใช้ PhysicsComponent

```java
@Override
protected void initInput() {
    FXGL.onKey(KeyCode.A, "Move Left", () -> getPlayer().translateX(-SPEED));
    FXGL.onKey(KeyCode.D, "Move Right", () -> getPlayer().translateX(SPEED));
    FXGL.onKey(KeyCode.W, "Move Up", () -> getPlayer().translateY(-SPEED));
    FXGL.onKey(KeyCode.S, "Move Down", () -> getPlayer().translateY(SPEED));
}
```

การจัดการ Input กับ Physics:

เมื่อใช้ PhysicsComponent การเคลื่อนไหวจะถูกควบคุมด้วยความเร็ว

ต้องใช้ UserAction สำหรับการควบคุมแบบละเอียดมากขึ้น

Override เมธอด:

onActionBegin() - เริ่มการเคลื่อนไหว (กำหนดความเร็วเริ่มต้น)

onActionEnd() - หยุดการเคลื่อนไหว (กำหนดความเร็วเป็นศูนย์)

onAction() - อัปเดตการเคลื่อนไหวในทุกเฟรม (ถ้าจำเป็น) ขณะที่กดค้างไว้

```
FXGL.getInput().addAction(new UserAction("Move Left") {
    @Override
    protected void onActionBegin() {
        getPlayer().getComponent(PhysicsComponent.class).setVelocityX(-SPEED));
    }
    @Override
    protected void onActionEnd() {
        getPlayer().getComponent(PhysicsComponent.class).setVelocityX(0));
    }
}, KeyCode.A);
```

ฟิสิกส์ (Physics)

การใช้ PhysicsComponent:

ใช้ PhysicsComponent สำหรับควบคุม rigid body

Rigid bodies สามารถชนและเด้งหรือหยุด แต่ไม่สามารถทะลุผ่านกันได้

ใช้กับกำแพง, พื้น, อุปสรรค และผู้เล่น

ค่าที่ใช้ด้านล่างเป็นค่าเริ่มต้นที่ดีสำหรับเกม 2D ทั่วไป

```
@Spawns("Player")
public Entity spawnPlayer(SpawnData data) {
    PhysicsComponent physics = new PhysicsComponent();
    physics.setFixtureDef(new FixtureDef().friction(0).density(0.1f));
    BodyDef bd = new BodyDef();
    bd.setFixedRotation(true);
    bd.setType(BodyType.DYNAMIC);
    physics.setBodyDef(bd);
    return FXGL.entityBuilder(data)
        .type(EntityType.PLAYER)
        .viewWithBBox("player.png")
        .with(physics)
        .build();
```

```java
@Override
protected void initPhysics() {
    FXGL.getPhysicsWorld().setGravity(0, 0);
}
```

การตั้งค่าแรงโน้มถ่วง (Gravity):
ใช้ setGravity(x, y) เพื่อตั้งค่าแรงโน้มถ่วงในทิศทางต่างๆ
ใช้แรงโน้มถ่วงเป็นศูนย์สำหรับเกมมุมมองจากด้านบน (top-down)
ใช้แรงโน้มถ่วงในแกน y สำหรับเกมแพลตฟอร์ม (platformers)
ค่าที่เหมาะสมอาจต้องทดลองหลายครั้ง

```java
@Spawns("Pill")
public Entity spawnPill(SpawnData data) {
    return FXGL.entityBuilder(data)
        .type(EntityType.PILL)
        .view("pill.png")
        // ทำให้ hit box เล็กกว่าเล็กน้อยเพื่อให้พอดีกับส่วนที่มองเห็นได้ของภาพ
        .bbox(new HitBox("PILL_HIT_BOX", new Point2D(5, 5), BoundingShape.box(9, 9)))
        .collidable()
        .build();
}
```

อย่าใช้ PhysicsComponent กับทุกอย่าง:

สำหรับวัตถุประเภทอื่นๆ (เช่น ศัตรู, กระสุน, ไอเทม) อาจไม่จำเป็นต้องใช้ PhysicsComponent

CollidableComponents ไม่ได้รับอิทธิพลจากฟิสิกส์และสามารถตรวจสอบการชน (วัตถุทับซ้อนกัน)