
JSGLib Tutorial 3: Ghosts

In this tutorial we create a little game to demonstrate multi-threaded use and further key input handling. A knight is trying to collect gold bars while avoiding ghosts.

Program.java

First, we create *Program.java* in *ghosts*:

```
package ghosts;
import jsplib.*;

public class Program {
    public static void main(String[] args) {
        Stage stage = new Stage("Ghosts", 800, 600, "forest.jpg");
    }
}
```

Knight.java

Next, we create the *Knight* class and derive it from *StageObject*:

```
package ghosts;
import jsplib.*;

public class Knight extends StageObject {

    public Knight(Stage s, int x, int y) {
        super(s, "Knight", x, y, "knight.png", 30);
    }

    public void act() {
        if (stage.isKeyPressed("DOWN"))
            changeY(4);
        if (stage.isKeyPressed("UP"))
            changeY(-4);
        if (stage.isKeyPressed("RIGHT"))
            changeX(4);
        if (stage.isKeyPressed("LEFT"))
            changeX(-4);
    }
}
```

Note that this object gets an id *Knight* (2nd argument for super) because we need to query it later on in other threads.

We override method *act* to check the arrow keys and move the object around. *changeX* and *changeY* add the given value to the current one. There is no range check needed. The object can move out of side but not further. *act* will be called every 20 ms by default.

Create a knight in *main*:

JSGLib Tutorial 3: Ghosts

```
public static void main(String[] args) {
    Stage stage = new Stage("Ghosts", 800, 600, "forest.jpg");
    new Knight(stage, 200, 200);
}
```

It's not working yet because we need to „start“ the stage to get a thread for each stage object:

```
public static void main(String[] args) {
    Stage stage = new Stage("Ghosts", 800, 600, "forest.jpg");
    new Knight(stage, 200, 200);
    stage.start();
}
```

It doesn't matter if the object is added before or after *start* is called. Each thread executes *StageObject.run* which loops until *Stage.stop* is called somewhere and calls *StageObject.act* every *StageObject.actDelay* milliseconds (default 20 ms). Usually you override the *act* method but you can also override *run* itself if you so desire.

Ghost.java

```
package ghosts;
import jsplib.*;

public class Ghost extends StageObject {

    public Ghost(Stage s, int x, int y) {
        super(s, x, y, "ghost.png", 30);
        setRotationStyle(StageObject.NONE);
        setRotation(Tools.rand(0, 360));
    }

    public void act() {
        move(4);
        if (hasHitBoundary())
            bounceOffBoundary();
    }

}
```

Ghost is also derived from *StageObject*. *setRotation* sets an angle directly. *setRotationStyle(NONE)* will prevent the image from being rotated.

act is overridden to move forward. *hasHitBoundary* returns true if object is out of side and *bounceOffBoundary* will adjust the angle to „bounce off“ the boundary.

Then we create 15 ghosts in the same spot in *main*:

```
public static void main(String[] args) {
    Stage stage = new Stage("Ghosts", 800, 600, "forest.jpg");
    new Knight(stage, 200, 200);
}
```

JSGLib Tutorial 3: Ghosts

```
        for (int i = 0; i < 15; i++)
            new Ghost(stage, 600, 400) ;
        stage.start();
    }
```

When a ghost and the knight collide the game is supposed to be over so complete *Ghost.act* with the instructions written in bold:

```
public void act() {
    move(4);
    if (hasHitBoundary())
        bounceOffBoundary();
    if (touches(stage.getObject("Knight"))) {
        new StageObject(stage, 400, 300, "gameover.png");
        stage.stop();
    }
}
```

getObject returns an object. You need to set an id for this object either when it's created or later on with *setId(String id)*. *Stage.stop* will cause all threads to terminate.

Goldbar.java

Now we add a gold bar that has to be collected by the knight:

```
package ghosts;
import jsplib.*;

public class Goldbar extends StageObject {

    public Goldbar(Stage s, int x, int y) {
        super(s, x, y, "goldbar.png", 20);
    }

    public void act() {
        if (touches(stage.getObject("Knight"))) {
            moveTo(Tools.rand(50, 750), Tools.rand(50, 550));
        }
    }
}
```

We override *act* to reposition the gold bar when it's touched by the knight. Alternatively, you could give this object an id *Goldbar* and check in class *Knight* if both collide.

In *main* we create it at the same position as the ghosts:

```
new Knight(stage, 200, 200);
new Goldbar(stage, 600, 400);
for (int i = 0; i < 15; i++)
    new Ghost(stage, 600, 400);
```

Time to Score

Finally, we count how many gold bars were collected before the game ends. Therefore we add a mixed variable to *stage* in *main*:

```
...
for (int i = 0; i < 15; i++)
    new Ghost(stage, 600, 400);

stage.addVariable("Score", 0);

stage.start();
...
```

And increase the counter in *Goldbar.java* when knight and gold bar touch:

```
public void act() {
    if (touches(stage.getObject("Knight"))) {
        moveTo(Tools.rand(50, 750), Tools.rand(50, 550));
        stage.changeVariable("Score", 1);
    }
}
```

A mixed variable can hold a *String*, *int* or *double*. *changeVariable* changes the current value by adding the given value. *setVariable* sets a value directly. A value can be queried by *getVariableAsInt*, *getVariableAsDouble* or *getVariableAsString*.