

# Ping Pong via Internet

Mattias Larsson

## Sammanfattning

Programmet är ett klassiskt ping pong-spel som kan spelas över internet eller lokalt mellan två datorer. Ping pong är ett av de äldsta datorspelen. Spelare 1 befinner sig på vänster sida av skärmen och spelare 2 på höger sida. Målet är att få bollen förbi motståndaren för att få poäng. Bollen studsar på alla ytor den kolliderar med. Spelet slutar när servern stänger av spelet, det finns inget sätt att vinna. Spelarna bestämmer själva när spelet är färdigt.

En spelare startar servern och spelar som spelare 1 (Server.java), medan spelare 2 (Client.java) ansluter till servern.

Jag har tidigare arbetat med många spelprojekt i andra kurser och har länge varit nyfiken på hur man utvecklar ett multiplayer-spel. Jag ville hålla spelet enkelt för att fokusera på kommunikationen mellan servern och klienten. Den största utmaningen med detta projekt var att göra nätverkskopplingen. Jag behövde också se till att spelet kändes bra för båda spelare som var anslutna, t.ex. att det inte gick för fort och att tiden det tog för spelet att reagera på den anslutande spelarens tangenttryck var rimligt.

# Framtagande

## Planering

Jag bestämde mig för att skapa ett nätverksbaserat ping pong spel där nätverkskopplingen var lik den tidigare chattapplikationen jag gjorde i en av uppgifterna. Jag inspirerades av flera exempel av ping pong spel som jag hittade online via youtube och github. Specifikt så följde jag denna youtube video för att lära mig om att koda ping pong.

<https://www.youtube.com/watch?v=oLirZqJFKPE>

Min plan var att börja utveckla spelet och sedan använda mig av strukturen som jag gjorde för chattapplikationen i en föregående uppgift. Jag ville ha en klass som kördes från spelaren som skapade servern (Server-klassen) och en annan som kördes från den anslutande spelaren (Client-klassen).

## Inledande implementering av spelet

Jag började med att göra ping pong som bara kunde spelas på en dator. Spelet är ganska simpelt. Två spelare styr varsin paddel upp och ner som kolliderar med en boll. Om bollen åker förbi motståndarens paddel får man en poäng. Första versionen av spelet spelade man på samma dator, spelare 1 använde W och S för att styra den vänstra paddeln och spelare två använde UPP och NER pilarna för att styra den högra.

## Utveckling av nätverksfunktionalitet

Efter det så började utvecklingen av servern, jag använde mig av koden för servern och klienten från chatt-applikationen. Jag började med att modifiera dessa klasser för att skicka inputs till servern från spelare 2 och sedan skicka tillbaka spelets grafik tillbaka till klienten som målar upp en kopia av vad servern har ritat på skärmen och hur många poäng spelarna har.

Servern hanterar speldata och skicka uppdateringarna av spelet till klienten

Klienten skickar inputs till servern som flyttar på klienten (spelare 2)

Servern processar spelet och uppdaterar spelarnas positioner poäng och skickar tillbaka information till klienten så att klienten visar samma spel som servern.

I början försökte jag skicka inputs som objekt men insåg att det var ineffektivt och man kunde göra det enklare. Istället skickas ett heltal som representerar olika inputs.

0 om man klickar på ner tangenten (KeyCode.DOWN)

1 om man släpper ner tangenten (KeyCode.DOWN)

2 om man klickar på upp tangenten (KeyCode.UP)

3 om man släpper upp tangenten (KeyCode.UP)

## Förbättringar av spelkänslan

Efter dessa steg så började jag testa spelet mycket för att förbättra användarupplevelsen. En av dessa problem som jag hittade under testningen var att spelet hade för hög framerate ifall man minimerade spelfönstret, vilket gjorde att spelet spelades för snabbt. För att lösa detta implementerade jag en enkel begränsning av framerate vilket gjorde så att spelet kändes bättre att spela.

Jag implementerade också en scoreboard-klass som visar spelets poäng. Det finns ingen maximal poäng för att vinna, spelarna bestämmer själva när spelet är klart.

## Avslutande justeringar

Den sista fasen av utvecklingen fokuserade främst på att fixa indenteringar, byta namn på variabler och metoder samt göra justeringar i kodstrukturen. Jag granskade även variablerna för att säkerställa att de hade korrekta access modifiers. Det sista som implementerades var popup-aviseringar för att informera användaren om anslutningsstatus och när servern har startat. Detta gör att användaren tydligare kan se programmets status när servern skapas och ansluts. Det finns även popup-aviseringar för exempelvis när servern bestämmer sig för att starta spelet medan en spelare fortfarande är ansluten. Nedan följer några exempel från server-klassen där dessa används. Det finns även alerts för klienten.

```
showAlert(AlertType.ERROR, "Invalid Port", "Port number must be between 1 and 65535.");  
showAlert(AlertType.ERROR, "Port Unavailable", "Port " + inputPort + " is already in use.");  
showAlert(AlertType.ERROR, "Player 2 Not Connected!", "Game is starting without player  
2.");
```

# Form

## Server Class

### showGameWindow(primaryStage)

Denna funktion visar spelet för den som startar servern och spelar den första spelaren. Skapar en canvas med specifik bredd och höjd och hämtar dess graphics context. En scen skapas med scenens bredd och höjd. Spelet initieras genom att anropa initializeGame(). Händelsehanterare sätts för tangenttryckningar och när man släpper för att hantera rörelsen av spelaren. En animations-timer skapas som uppdaterar och ritar spelet med ett tidsintervall som är begränsat så att det inte sker för många uppdateringar för snabbt. Slutligen sätts en händelsehanterare för att hantera när fönstret stängs, vilket stänger ned systemet och avslutar programmet.

```
FUNCTION showGameWindow(primaryStage):

    CREATE canvas WITH WIDTH and HEIGHT
    SET gc TO graphics context of canvas

    CREATE root group
    ADD canvas TO root children

    CREATE scene WITH root, WIDTH, and HEIGHT
    SET primaryStage scene TO scene
    SHOW primaryStage

    CALL initializeGame()

    CREATE activeKeys AS new concurrent key set

    SET scene key pressed event handler TO handleKeyPress
    SET scene key released event handler TO handleKeyRelease

    SET frameInterval TO 100000000 / 60

    CREATE new AnimationTimer:
        DEFINE lastUpdate as 0

    OVERRIDE handle(now):
        IF (now - lastUpdate >= frameInterval):
            CALL updateGame()
            CALL draw(gc)
            SET lastUpdate TO now
    START AnimationTimer

    SET primaryStage close request event handler TO:
        CALL shutdown()
        EXIT platform
        TERMINATE system
    END FUNCTION
```

## run()

Denna funktion tar in inputs från klienten. Försöker skapa "out" och "in" object streams från klientens streams. Spelarens namn sätts i Scoreboard objektet och skickas ut. En loop körs så länge messages tas emot. Om meddelandet är en Integer, behandlas olika värden för att ställa in flaggor för key presses. Om meddelandet är en String är det klientens namn och det sätts som spelarens namn i scoreBoard. Vid exceptions skrivs ett felmeddelande ut och vid avslutning stängs anslutningen ned.

FUNCTION run():

TRY:

CREATE out AS new ObjectOutputStream FROM client output stream  
CREATE in AS new ObjectInputStream FROM client input stream

CALL scoreBoard.setP1Name(playerName)  
CALL broadcast(playerName)

LOOP while message is received:  
  READ message from input stream  
  IF message is an Integer:  
    SET val TO Integer value of message

    IF val is 0:  
      SET clientPressedDown TO true  
    ELSE IF val is 1:  
      SET clientPressedDown TO false  
    ELSE IF val is 2:  
      SET clientPressedUp TO true  
    ELSE IF val is 3:  
      SET clientPressedUp TO false

  ELSE:  
    SET s TO String value of message  
    CALL scoreBoard.setP2Name(s)

CATCH ClassNotFoundException OR IOException AS e:  
  PRINT "Connection error:"

FINALLY:  
  CALL shutdown()

END FUNCTION

CLASS GameState IMPLEMENTS Serializable:

  CONSTANT serialVersionUID = 1L

  PROPERTY ballX, ballY AS double

  PROPERTY player1Y, player2Y AS double

## GameState

En intern klass som representerar spelets "Game state" med bollens och spelarnas koordinater som properties. En constructor initierar dessa värden. En funktion uppdaterar statuset med nya värden.

```
FUNCTION GameState(ballX, ballY, player1Y, player2Y):
```

```
    SET this.ballX TO ballX
```

```
    SET this.ballY TO ballY
```

```
    SET this.player1Y TO player1Y
```

```
    SET this.player2Y TO player2Y
```

```
END FUNCTION
```

```
FUNCTION updateGameState(gameState):
```

```
    SET this.player1Y TO gameState.player1Y
```

```
    SET this.player2Y TO gameState.player2Y
```

```
    SET this.ballX TO gameState.ballX
```

```
    SET this.ballY TO gameState.ballY
```

```
END FUNCTION
```

```
END CLASS
```

## Client Class

### run()

Försöker skapa en socket med specifik IP och port, samt object streams för kommunikation med servern. Spelarens namn sänds till servern och en framgångsrik anslutningsmeddelande visas. En loop hanterar mottagna messages så länge spelet inte är avslutat. Om ett GameState objekt tas emot, uppdateras spelet med de nya värdena. Om en Integer tas emot, uppdateras scoreBoard med rätt spelares score. Om en String tas emot, hanteras starten av spelet eller utskrift av meddelandet. Vid exceptions hanteras fel och anslutningen stängs ned.

FUNCTION run():

TRY:

```
CREATE client socket WITH ipField text AND parsed portField text
CREATE out AS new ObjectOutputStream FROM client's output stream
CREATE in AS new ObjectInputStream FROM client's input stream
```

```
CALL out.writeObject(playerName)
CALL out.flush()
```

```
CALL showAlert WITH AlertType.INFORMATION, "Connection Status", "Connection to server successful!"
```

```
LOOP while not done AND message is received:
  READ message from input stream
```

```
IF message is an instance of Server.GameState:
  SET gameState TO Server.GameState value of message
  CALL Platform.runLater TO updateGameState WITH gameState
```

```
ELSE IF message is an instance of Integer:
  PARSE message to integer i
```

```
IF firstPlayerScore is true:
  CALL scoreBoard.setP1Score WITH i
  SET firstPlayerScore TO false
```

```
ELSE:
  CALL scoreBoard.setP2Score WITH i
  SET firstPlayerScore TO true
```

```
ELSE IF message is an instance of String:
  IF message is "StartGame":
    CALL startGame()
  ELSE:
    PRINT message
    CALL scoreBoard.setP1Name WITH message
```

```
CATCH NumberFormatException AS e:
  CALL showAlert WITH AlertType.ERROR, "Invalid Port", "Please enter a valid port number."
```

```
CATCH IOException AS e:
  PRINT "Failed to connect to server ip: " + ipField text + " Port: " + portField text
  CALL showAlert WITH AlertType.ERROR, "Connection Error", "Failed to connect to the server."
```

```
CATCH Exception AS e:  
  IF not done:  
    PRINT stack trace of e  
  CALL shutdown()
```

```
END FUNCTION
```

### **sendInputToServer(input)**

Synkroniserar sändning av input till servern om klient är ansluten och streams inte är stängda. Annars skrivs ett felmeddelande ut.

```
FUNCTION sendInputToServer(input):  
  SYNCHRONIZED:  
    IF client is not null AND client is not closed AND out is not null:  
      CALL out.writeObject WITH input  
      CALL out.flush()  
    ELSE:  
      PRINT "Cannot send input to server: Socket is closed or output stream is null"  
  
END FUNCTION
```



## **startGame()**

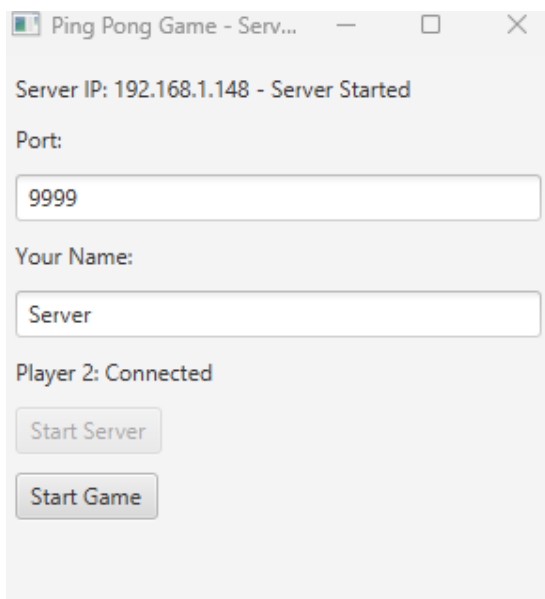
Startar spelet om det inte redan är startat. Använder Platform.runLater för att skapa och visa ett nytt fönster för spelet med en canvas och graphics context. Event handlers sätts för key pressed och key released. Om en av dessa key events triggas så skickas input till server som en Integer. En AnimationTimer skapas för att rita spelet kontinuerligt baserat på information som skickas från servern.

```
FUNCTION startGame():  
  IF gameStarted is false:  
    SET gameStarted TO true  
  
    CALL Platform.runLater WITH:  
      CREATE primaryStage AS new Stage  
      SET primaryStage title TO "Ping Pong Game - Client"  
  
      CREATE canvas WITH WIDTH and HEIGHT  
      SET gc TO graphics context of canvas  
  
      CREATE root group  
      ADD canvas TO root children  
  
      CREATE scene WITH root, WIDTH, and HEIGHT  
      SET primaryStage scene TO scene  
      SHOW primaryStage  
  
      SET scene key pressed event handler TO handleKeyPress  
      SET scene key released event handler TO handleKeyRelease  
  
      CREATE new AnimationTimer:  
        OVERRIDE handle(now):  
          CALL draw WITH gc  
      START AnimationTimer  
  
      SET primaryStage close request event handler TO:  
        CALL shutdown()  
        CALL Platform.exit()  
        CALL System.exit(0)  
END FUNCTION
```

# Funktion

När man startar servern, visas ett fönster med följande komponenter

- Server IP: Visar IP-adressen för servern.
- Port: Textfält där du kan ange porten som servern ska lyssna på (standard är 9999).
- Ditt namn: Textfält där du kan ange ditt namn som kommer att visas som spelarens namn i spelet.
- Spelare 2-status: Visar om spelare 2 har anslutit eller inte (inledningsvis "Not Connected").
- Start Server: En knapp för att starta servern och börja lyssna på inkommande anslutningar.
- Start Game: En knapp som blir aktiv när spelare 2 har anslutit sig. Används för att starta spelet.
- När du klickar på "Start Server" börjar servern lyssna på inkommande anslutningar.
- Serverns status uppdateras och visar IP-adressen och att servern har startat.

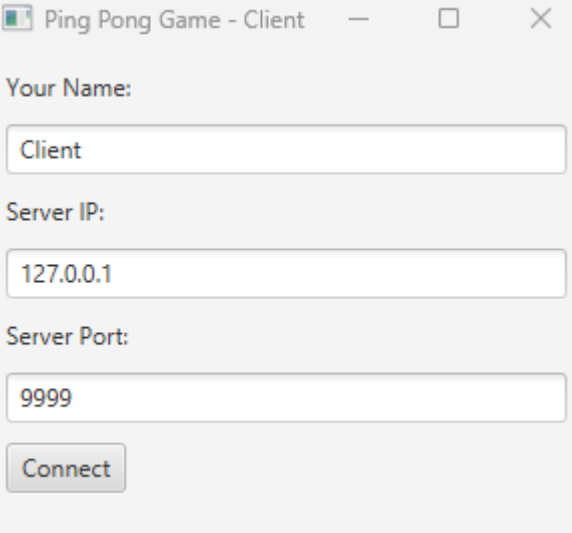


*Den här bilden visar hur det ser ut att starta servern*

## Ansluta med Klienten

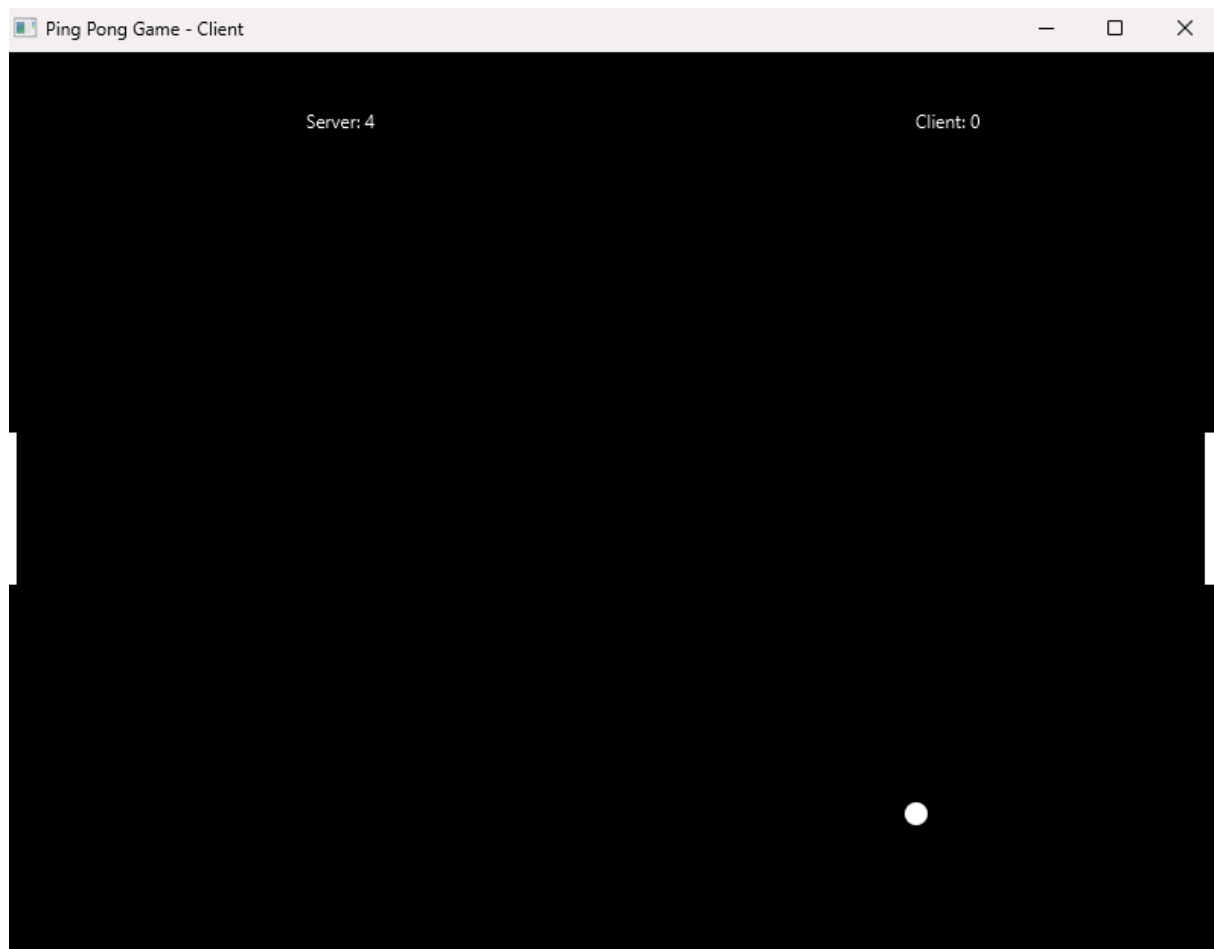
När du startar klienten visas ett fönster med följande komponenter

- Ditt namn: Textfält där du kan ange ditt namn.
- Server IP: Textfält där du anger servers IP-adress (standard är 127.0.0.1 för lokal anslutning).
- Server Port: Textfält där du anger servers port (standard är 9999).
- Connect: En knapp för att ansluta till servern.
- När du klickar på "Connect" försöker klienten ansluta till servern med den angivna IP-adressen och porten. Om anslutningen lyckas, skickas spelarens namn till servern och anslutningsstatusen uppdateras.



The screenshot shows a window titled "Ping Pong Game - Client". Inside the window, there are three text input fields and one button. The first field is labeled "Your Name:" and contains the text "Client". The second field is labeled "Server IP:" and contains the text "127.0.0.1". The third field is labeled "Server Port:" and contains the text "9999". Below these fields is a button labeled "Connect".

*Den här bilden visar hur gränssnittet ser ut för klienten som ska ansluta till servern*



*Den här bilden visar spelet*

Dessa är kommandon för att starta båda program, man behöver länka med javafx.

Server

```
java --module-path "E:\javafx-sdk-22.0.1\lib" --add-modules javafx.controls,javafx.fxml -jar server.jar
```

```
PS E:\IPROG\Prov\src> java --module-path "E:\javafx-sdk-22.0.1\lib" --add-modules javafx.controls,javafx.fxml -jar server.jar
```

Client

```
java --module-path "E:\javafx-sdk-22.0.1\lib" --add-modules javafx.controls,javafx.fxml -jar client.jar
```

```
PS E:\IPROG\Prov\src> java --module-path "E:\javafx-sdk-22.0.1\lib" --add-modules javafx.controls,javafx.fxml -jar client.jar
```