# Metoda wytwórcza

```
package com.company;

import java.io.IOException;

public abstract class TcpFactory implements Runnable {

    public void run() {
        Runnable task = null;
        try {
            task = createTask();
        } catch (IOException e) {
            e.printStackTrace();
        }
        assert task != null;
        task.run();
    }
    public abstract Runnable createTask() throws IOException;
}
```

```java
package com.company;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class TcpClient extends TcpFactory {
    private static final int POOL_SIZE = 10;
    private static final int RUNS = 100;

    void fire() {
      ExecutorService executor =
Executors.newFixedThreadPool(POOL_SIZE);
      for ( int i = 0; i < RUNS; i++ ) {
          executor.execute(new ClientTask(i));
      }
      executor.shutdown();
    }

    public static final void main(String[] args)  {
      new TcpClient().fire();
    }


    @Override
    public Runnable createTask() {
        return new TcpClient();
    }
}
```

```java
package com.company;

import java.io.IOException;
import java.net.ServerSocket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class TcpServer extends TcpFactory {
    public static final int PORT = 8205;
    private static final int POOL_SIZE = 10;
    private ServerSocket ss;
    private ExecutorService executor;


    public TcpServer() throws IOException {
      ss = new ServerSocket(PORT);
      executor = Executors.newFixedThreadPool(POOL_SIZE);
    }

    public void listen() throws IOException {
      System.out.println(this + " up and running");
      while ( true ) {
          executor.execute(new ServerTask(ss.accept()));
      }
    }

    public static final void main(String[] args) throws IOException {
      new TcpServer().listen();
    }

    @Override
    public String toString() {
      return "TcpServer [port=" + ss.getLocalPort() + "]";
    }

    @Override
    public Runnable createTask() throws IOException {
        return new TcpServer();
    }
}
```

# Fabryka abstrakcji

```java
package com.company;

import java.io.IOException;

public interface TcpFactory {
    TcpClient createTcpClient();
    TcpServer createTcpServer() throws IOException;
}
```

```java
package com.company;

public class TcpClientFactory implements TcpFactory {
    @Override
    public TcpClient createTcpClient() {
        return new TcpClient();
    }

    @Override
    public TcpServer createTcpServer() {
        return null;
    }
}
```

```java
package com.company;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class TcpClient extends TcpClientFactory {
    private static final int POOL_SIZE = 10;
    private static final int RUNS = 100;

    void fire() {
      ExecutorService executor =
Executors.newFixedThreadPool(POOL_SIZE);
      for ( int i = 0; i < RUNS; i++ ) {
          executor.execute(new ClientTask(i));
      }
      executor.shutdown();
    }

    public static final void main(String[] args)  {
      new TcpClient().fire();
    }

}
```

```java
package com.company;

import java.io.IOException;

public class TcpServerFactory implements TcpFactory {
    @Override
    public TcpClient createTcpClient() {
        return null;
    }

    @Override
    public TcpServer createTcpServer() throws IOException {
        return new TcpServer();
    }
}
```

```java
package com.company;

import java.io.IOException;
import java.net.ServerSocket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class TcpServer extends TcpServerFactory {
    public static final int PORT = 8205;
    private static final int POOL_SIZE = 10;
    private ServerSocket ss;
    private ExecutorService executor;


    public TcpServer() throws IOException {
      ss = new ServerSocket(PORT);
      executor = Executors.newFixedThreadPool(POOL_SIZE);
    }

    public void listen() throws IOException {
      System.out.println(this + " up and running");
      while ( true ) {
          executor.execute(new ServerTask(ss.accept()));
      }
    }

    public static final void main(String[] args) throws IOException {
      new TcpServer().listen();
    }

    @Override
    public String toString() {
      return "TcpServer [port=" + ss.getLocalPort() + "]";
    }

}
```