# Zaawansowane programowanie obiektowe
# Lab. 11
# (Programowanie sieciowe w Javie; Scala)

1. KOMUNIKACJA KLIENT-SERWER (1,5 pkt)
   Napisz dwa programy, symulujące wyścig kolarski (tym razem: szosowy :-), łączące się przez gniazdo (ang. *socket*) i działające w następujący sposób:
   - KLIENT: symuluje w konsoli przejazd jednego kolarza o podanym przy wywołaniu imieniu i nazwisku (np. „Ryszard Szurkowski", możesz wziąć przykładowe nazwiska spod https://pl.wikipedia.org/wiki/Kategoria:Zwyci%C4%99zcy_Wy%C5%9Bcigu_Pokoju) w sposób, który ukazany jest na filmiku: http://szgrabowski.kis.p.lodz.pl/zpo19/race.mp4 (poczekaj, pierwsze 3-4 sek. nic się nie dzieje). Uwaga: parametry ruchu na filmie są nieco inne niż podane niżej w treści zadania.

     Przydatne napisy:

```
String route = "S                    |        |         |                |           |      M";
String bike = "o&o";
```

     Wyjaśnienie: S – pozycja startowa, M – meta, kreski pionowe – lotne finisze.
     Kolarz osiąga dany punkt (metę lub lotny finisz) wtedy, gdy „najeżdża" na niego przednim kołem roweru. Rower przesuwa się o jednostkę (znak graficzny) co losowy czas od 30ms do 150ms, losowany osobno dla każdego „ruchu", z dokładnością do 10ms.
     Wskazówka: do symulacji ruchu wykorzystaj znak powrotu karetki (\r).

   - SERWER: czeka aż połączy się z nim 3 klientów (czyli 3 kolarzy gotowych do wyścigu), a następnie wysyła wszystkim komunikat „Start!" Gdy klient odbierze ten komunikat – zaczyna jechać. Gdy osiąga lotny finisz lub metę, wysyła o tym informację do serwera. Gdy pierwszy z takich komunikatów dla danego lotnego finiszu dotrze do serwera, wtedy serwer wypisuje komunikat: X wygrał lotny finisz nr Y (np. „Lech Piasecki wygrał lotny finisz nr 3"). Podobnie z osiągnięciem mety (np. „Ryszard Szurkowski pierwszy na mecie!"). Ponadto, gdy dany zawodnik (każdy z trzech) dojedzie do mety, wtedy serwer przesyła mu numer zajętego miejsca. Zaraz po otrzymaniu tej wiadomości **klient** wypisuje informację o swoim miejscu, np. „Lech Piasecki - nr 2 na mecie".

2. SCALA, ROZGRZEWKA (0,5 pkt)
   Dany jest string postaci "-3 + 4 - 1 + 1 + 12 - 5 + 6", tj. liczby całkowite, między nimi operator + lub -, do tego rozdzielające spacje. Policz wartość tego wyrażenia korzystając z mechanizmu pasowania wzorców (ang. *pattern matching*).
   Jeśli zamiast poprawnego operatora jest inny napis (np. # lub ABC), to wtedy należy rzucić stosowny wyjątek.

3. SCALA, foldLeft (0,5 pkt)
   Napisz funkcję w Scali, sprawdzającą czy przekazane jako string wyrażenie nawiasowe jest poprawne. W rozwiązaniu wykorzystaj mechanizm pasowania wzorców (*pattern matching*) i metodę foldLeft. (Dla znawców Pythona: foldLeft przypomina funkcję functools.reduce)

   Przykłady testów:

```
assert(checkBalancedParens("))((") == false)
assert(checkBalancedParens("(()(ab)c)"))
assert(checkBalancedParens("(((()))") == false)
assert(checkBalancedParens("(()))(") == false)
assert(checkBalancedParens("(aa(b)()c)d"))
```

4. SCALA, HASŁO (1 pkt)
   Pewna aplikacja używa logowania przy użyciu hasła. Wiadomo, że nie każdy string powinien być dopuszczony jako hasło. Programiści tworzący tę aplikację nie są jednak jeszcze zgodni, co do konkretnych kryteriów, jakie musi spełniać hasło.

   Napisz obiekt (object Password) z funkcjonalnością walidowania hasła w taki sposób, aby łatwo było dostosować przyjęte kryteria. Zakładamy, że ostateczna definicja poprawnego hasła korzysta z podzbioru następujących wymogów:
   - hasło ma minimalną długość min_len (dowolny parametr),
   - hasło ma maksymalną długość max_len (dowolny parametr),
   - zawiera co najmniej 1 wielką literę,
   - zawiera co najmniej 1 małą literę,
   - zawiera co najmniej 1 cyfrę,
   - zawiera co najmniej 2 cyfry.

   Wskazówki:
   1. W Scali argumentami funkcji/metod mogą być inne funkcje – wykorzystaj ten mechanizm.
   2. Parametry min_len i max_len NIE powinny być polami obiektu. Jak inaczej je przekazać, skoro pozostałe funkcje mają tylko 1 parametr (tj. string)? Wykorzystaj mechanizm rozwijania funkcji (*currying*).