Cel ćwiczenia

Celem ćwiczeń zawartych w projekcie jest:

- zapoznanie się z funkcjonalnością podstawowych komend rozproszonego systemu kontroli wersji Git,
- wprowadzenie do wykorzystania platformy GitHub,
- zrozumienie znaczenia kontroli wersji i potrzeby wykorzystywania systemu kontroli wersji w pracy indywidualnej i zespołowej,
- zapoznanie się z powszechnie stosowanymi konwencjami dotyczącymi organizacji struktury projektów Java oraz kodu języka Java (m. in. formatowanie i nazewnictwo),
- przegląd podstawowych technik mających na celu poprawienie i utrzymanie wysokiej jakość kodu w języku Java,
- kolejnym istotnym elementem laboratoriów jest nauka wykorzystania zaawansowanego środowiska programistycznego IDE (przykładowe wyjaśnienia dotyczą środowiska Eclipse lub IntelliJ) do automatycznej generacji kodu oraz refaktoryzacji.

Projekt pio_git_rhymers to biblioteka dostarczająca egzotyczne warianty *wyliczanek*, które służą przeprowadzaniu rozrywek skomplikowanych gier całkowito liczbowych. O ile działanie bazowej klasy wyliczankowej DefaultCountingOutRhymer w swoim działaniu przypomina stos:

- FIFORhymer – dostarczający pod klasycznym interfejsem wyliczankę opartą na podejściu kolejki First In First Out
- HanoiRhymer – na którym nie jest możliwe zgłoszenie (countIn) liczby większej niż bieżąca (peekaboo).

Polecenia laboratorium będą dotyczyły programowania w języku Java. Potrzebne będzie środowisko dla programistów (JDK – Java Development Kit) oraz zintegrowana platforma programistyczna (np. Eclipse lub IntelliJ) z zainstalowaną wtyczką do obsługi narzędzia Maven (np. m2eclipse).

Uwagi i zasady realizacji ćwiczeń

Oceniane będą: poprawność merytoryczna, czystość kodu, opisy commitów w git oraz terminowe oddanie danego fragmentu.

Higiena ogólna:

- Przeczytaj każdy rozdział instrukcji do końca przed rozpoczęciem jego realizacji.
- Wszędzie gdzie jest to napisane wykorzystuj narzędzia środowiska IDE.
- Polecenia gita wykonuj z konsoli (nie za pomocą środowiska IDE).
- Wykonanie każdego punktu nie powinno wprowadzać nowych błędów.
- W razie kłopotów korzystaj z pomocy prowadzącego.
- Alternatywne pomysły na rozwiązanie zadań (najlepiej) zgłoś prowadzącemu.
- Domyślnie (tj. o ile nie napisano inaczej) polecenia ćwiczeń stosuj w klasach w *src/main/java* (nie w plikach z testami).
- Pracując nad projektem dbaj o poprawność testów jednostkowych.
- Poprawiaj tylko to co jest w poleceniu konkretnego podpunktu.

Praca z systemem kontroli wersji:

- Wykonuj commit do każdego wykonanego punktu laboratorium.
- Komentarz przy commicie powinien zawierać punkt instrukcji. Oprócz numeru wykonanego punktu powinien zawierać krótki opis (najlepiej w postaci kluczowych sformułowań np. “1. Dodanie gitignore”).
- Przechowywanie folderów generowanych automatycznie przez narzędzia i środowisko IDE (np. *bin*, *target*, *.idea*) w repozytorium utrudnia prace i jest niewskazane. Dla wygody możesz skorzystać z pliku *.gitignore*.

Sposób oddania zadania:

- Przy oddaniu działu „Dodanie gitignore i praca na gałęziach” proszę o podanie w odowiedzi zadania „Laboratorium GitHub” linka prowadzącego bezpośrednio do repozytorium -SAMEGO linka, bez żadnych dodatkowych opisów, bez słowa link. Link musi po przeklejeniu go do przeglądarki prowadzić do repozytorium.
- W nowej linii dodanie wpisu: data – oddaję Dział I.
- Podczas oddawania kolejnych działów proszę o dodawanie w nowych liniach kolejnych wpisów z datą i numerem działu (np. data – oddaję Dział II:).
- Odpowiedzi na pytania umieść na WIKAMPIe w zadaniu “Laboratorium GitHub”. Wszystkie odpowiedzi umieść w jednym pliku .pdf i załącz go do odpowiedzi.
- Zrzuty ekranu umieść w pliku pdf, a nie jako treść odpowiedzi.
- Po wykonaniu każdego działu wrzuć plik z aktualnym stanem repozytorium do odpowiedniego zadania(zadanie GitHub_dzial_1, zadanie GitHub_dzial_2, zadanie GitHub_dzial_3)

Wyjściowo z zdania student otrzymuje ocenę 5. Każdy błąd (lub grupa powiązanych ze sobą błędów) obniża ocenę o 0.2 lub 0.1 w zależności od wagi błędu. W dziale III są zdania dodatkowe które jeżeli się wykona poprawnie to każdy z punktów podnosi ocenę o 0.1. Oddanie zadanie po terminie obniża ocenę o 0.5. Za każdy kolejny tydzień zwłoki ocena jest obniżana o kolejne 0.5 oceny.

Każdy student odpowiada za swoje repozytorium, czy jeżeli kolabolator wprowadził błędne zmiany a właściciel repozytorium je scalił to właściciel odpowiada za błędy w repozytorium.

Student ma możliwość poprawy jednego laboratorium w przypadku, kiedy to laboratorium zostało oddane w terminie. Poprawki można oddawać do końca 9 laboratorium. W przypadku, gdy po poprawce student dalej nie zaliczy, ma możliwość wykonania kolejnej poprawki, ale wtedy maksymalna ocena z zadania to 3. Drugą poprawkę można oddać do końca 14 laboratorium.

Przygotowanie repozytorium i projektu

- Jeżeli nie masz konta na platformie github, załóż je.
- Na platformie github zrób **fork** projektu (opcja dostępna po zalogowaniu). Link dostępny na wikampie. Odznacz opcję “**Copy the** master **branch only**”, żeby oprócz gałęzi **master** skopiować również gałąź **format**.
- Fork repozytorium projektu należy sklonować (np. *git clone*) i zaimportować do wybranego środowiska IDE (np. w Eclipse IDE wybierając *File → Import... → Maven → Existing Maven Projects*. Następnie należy wybrać katalog zawierający plik *pom.xml* jako *Root Directory* i kliknąć *Finish*).
- Zapoznaj się z projektem, jego strukturą, klasami. Uruchom aplikację demo *RhymersDemo*. Uruchom również testy jednostkowe: menu kontekstowe projektu *Run as ... → Maven test*. Zapoznaj się z testami jednostkowymi klasy *DefaultCountingOutRhymer* (znajdują się w *src/test/java*).

Dział I: Dodanie gitignore i praca na gałęziach

Przechowywanie folderów generowanych automatycznie przez narzędzia i IDE (np. *bin*, *target*, *.idea*) w repozytorium utrudnia prace i jest niewskazane. Pierwsze punkty z instrukcji (punkt 1. i 2.) należy wykonać na gałęzi master.

1. Do repozytorium dodaj plik *.gitignore* z odpowiednią zawartością, która zabezpieczy projekt przed nieumyślnym dodaniem niepożądanych plików i folderów. Wprowadź zmiany do repozytorium (polecenia *git add, git commit*) właściwie je nazywając (np. „1. dodanie pliku gitignore").

2. Popraw błędy związane z formatowaniem kodu źródłowego Java w klasie HanoiRhymer. Na zaznaczonym fragmencie kodu używaj klawisza **tab** lub kombinacji **shift+tab** do regulacji wcięć. Wprowadź zmiany do repozytorium.
   Wprowadź zmiany do repozytorium zdalnego (polecenie git push).

   Uwaga: od tej pory instrukcja pomija aspekt wprowadzania zmian do lokalnego repozytorium. To znaczy, że po każdym wykonanym punkcie (czyli 1, 2, 3..) należy zrobić commit, który zawiera nr. punktu oraz opis co zostało zrobione w tym punkcie.

3. Na platformie **github** przełącz się na gałąź **format**. Wyświetl listę commitów i odpowiedz na *pytanie*: jak się nazywał ostatni commit w tej gałęzi i które linie zostały w nim usunięte. Odpowiedzi na to i następne pytania umieść w pliku pdf. Wszystkie odpowiedzi umieść w jednym pliku .pdf i załącz go do odpowiedzi.

   W terminalu porównaj plik HanoiRhymer.java w gałęzi **master** i **format** (możesz użyć polecenia *git diff origin/format *Hanoi**). Dopóki będą jakieś różnice w pliku HanoiRhymer.java wykonaj ponownie punkt 2.

   W terminalu przełącz się na gałąź **master** i scal zmiany z gałęzi **format** do gałęzi **master** przy użyciu poleceń *git merge --squash* i *git commit*.
   Sprawdź, czy faktycznie wszystkie błędy formatowania zostały usunięte we wszystkich klasach (ogranicz się do folderu /src/main/java) uruchamiając automatyczne formatowanie (Source → Format lub ctrl+shift+f w elipse lub ctrl+alt+l w IntelliJ). Jeżeli nie, zgłoś problem prowadzącemu. Zweryfikuj działanie kombinacji klawiszy alt + ← oraz alt + →. Odpowiedź umieść w pliku pdf.

4. Utwórz gałąź (polecenie *git branch*) o nazwie **class-method-naming** na potrzeby realizacji zadania dotyczącego konwencji nazewniczej klas i metod. Przełącz się na gałąź **class-method-naming** (polecenie *git checkout*).
   Na gałęzi **class-method-naming** popraw błędy konwencji nazewniczej **klas i metod** kodu źródłowego Java we wszystkich klasach. Zrób osobny commit dla zmian nazw klas i osobny dla zmian nazw metod (np. „4.1 poprawa nazw klas" lub „4.2 zmiana konwencji nazewniczej metod"). Użyj do tego opcji Refactor → Rename (skrót klawiszowy alt+shift+r w eclipse lub Shitf+F6 w IntelliJ). Nie zmieniaj ułożenia metod ani logiki kodu - zmieniaj tylko nazwy. (Code Conventions for the Java TM Programming Language - rozdział 9)

5. Przełącz się na gałąź **master** i utwórz gałąź (polecenie *git branch*) o nazwie **fields-naming** i przełącz się na nią. Popraw błędy konwencji nazewniczej **pól klas** we wszystkich klasach i zrób commit.

Przełącz się na gałąź **master** i scal zmiany z gałęzi **fields-naming** do gałęzi **master** przy użyciu polecenia *git merge --ff-only*.

Gałąź **class-method-naming** nie wychodzi teraz ze szczytu gałęzi **master**. Przełącz się na gałąź **class-method-naming** i użyj polecenia *git rebase*, aby przenieść ją na szczyt gałęzi **master**. Przełącz się na gałąź **master** i scal zmiany z gałęzi **class-method-naming** do gałęzi **master** przy użyciu polecenia *git merge --no-ff*.

Wprowadź wszystkie zmiany do zdalnego repozyturium. Zrób printscreena z grafu commitów i dodaj go do pdf z odpowiedzią (proszę nie wklejać go w treść odpowiedzi). Printscreeen ma być wykonany z polecenia git lg, lub z grafu komitów z Inteligeii (nie z GitHub).

Pytanie: Wyjaśnij własnymi słowami różnicę działania wszystkich opcji polecenia *merge* używanych podczas realizacji zadań.
*Wykorzystaj narzędzie https://git-school.github.io/visualizing-git/ do zilustrowania odpowiedzi.
Wyjaśnij również zachowanie polecenia *rebase*.
Odpowiedzi na pytania dodaj do pliku pdf.

Dział II: Praca w podejściu feature branch

- Skonfiguruj ustawienia repozytorium tak, żeby dalsze zmiany do gałęzi master mogły być wprowadzane wyłącznie przy użyciu mechanizmu Pull Request: w ustawieniach gałęzi repozytorium (Settings -> Branches) dodaj odpowiednią regułę (Add branch protection rule) dla gałęzi master (Require pull request before merging + Require approvals).
- Skonfiguruj ustawienia repozytorium tak, żeby jedyną opcją dołączenia do gałęzi master był merge commit: w głównych ustawieniach repozytorium (Settings -> General, Pull Request) i pozostaw włączoną tylko opcję Allow merge commits.

Przygotowanie pracy zespołowej

- Zgłoś gotowość do pracy zespołowej prowadzącemu, aby ustalił kto będzie Twoim współpracownikiem (Collaborator).

- Dodaj Collaboratora jako kontrybutora do swojego repozytorium (*Settings* -> *Collaborators* -> *Add people*).
- Przy okazji, dodaj również prowadzącego jako kontrybutora do swojego repozytorium (będzie to potrzebne do ostatecznego zgłoszenia rozwiązania).
- W zakładce *Issues* na github utwórz nowy *Issue* z następującą treścią (na github czasami brakuje opcji tworzenia ISSUES, należy wtedy zaznaczyć checkbox feature ISSUES w ogólnych ustawieniach repoozytorium):

Stałe i annotacje

- Sklonuj repozytorium do którego zostałeś dodany i utwórz nowy branch poleceniem (polecenie git checkout -b constants)

6. Przejrzyj kod w poszukiwaniu literałów (napisy, liczby), które można by zastąpić deklaracjami stałych (np. w klasie DefaultCountingOutRhymer liczby -1 i 12). Wygeneruj odpowiednie stałe używając opcji Refactor → Extract Constant.
7. Ustaw modyfikator final przy niemutowalnych (nie zmieniających wartości) atrybutach klas.
8. Użyj annotacji @Override przy metodach tam gdzie jest to możliwe.

- Prześlij zmiany do zdalnego repozytorium (właściciela) – polecenie *git push origin* constants.
- Przejdź na główną stronę repozytorium i wybierz opcję New pull request.
- Jako base repository wybierz repozytorium właściciela i branch master, a jako head repository wybierz branch na którym pracowałeś, po czym kliknij Create pull request.
- Przejdź do strony z nowo utworzonym pull requestem i z kliknij zębatkę przy liście Reviewers, wybierz właściciela repozytorium jako recenzenta kodu (reviewer).
- Na tej samej stronie powiąż *pull request* z *Issue* (opcja Development panelu bocznego po prawej stronie).

- Do utworzonego *Issue* przypisz jako wykonawcę wyznaczonego współpracownika (Assigness po prawej stronie). Dodatkowo możesz go poprosić metodami tradycyjnymi o realizację w/w *Issue*.
- Jeżeli twój **collaborator** nie dokończył „u siebie" pracy nad poprzednimi rozdziałami oraz konfiguracją współpracy w ramach jego repozytorium (ustawienia repozytorium, przydzielenie collaboratora oraz wstawienie *Issue*) pomóż mu.

- Jeżeli **nie** zostałeś jeszcze przydzielony do pracy jako kolaborator nad *Issue* w innym repozytorium to możesz już wykonywać **Zadania równoległe do pracy zespołowej**.

- Jeżeli żaden **właściciel repozytorium** nie poprosił Cię o bycie współpracownikiem (**collaborator**), zgłoś problem do prowadzącego.
- Gdy **właściciel** skończy dodawanie **kontrybutora**, znajdź na swoim mailu i zaakceptuj zaproszenie do wspólnej pracy nad repozytorium.
- Wykonaj teraz tą pracę opisaną w ramach przydzielonego tobie *Issue* – Stałe i annotacje.
- Na platformie WIKAMP przedmiotu odnajdź aktywność test *Github - PR checkpoint*, wykonaj ją (hasło dostępu: review) i zapoznaj się ze zwrotną odpowiedzią.

- Jeżeli jeszcze **nie** otrzymałeś pull request-a od **kontrybutora** możesz już wykonywać **Zadania równoległe do pracy zespołowej** (o ile jeszcze tego nie zrobiłeś).

- Dokonaj przeglądu kodu napisanego przez kontrybutora i poproś o zmiany (jeżeli widzisz błędy wskazane w zwrotnej odpowiedzi testu *Github - PR checkpoint* lub dowolne inne) lub zaakceptuj zmiany i włącz je do brancha głównego (master). Powtarzaj ten krok dopóki w kodzie występują błędy (postaraj się, aby Twoje uwagi były precyzyjne). W razie problemów poproś o pomoc prowadzącego.

Zadania równoległe do pracy zespołowej

- Utwórz nową gałąź na potrzeby realizacji tematu: Accessory i hermetyzacja.

9. Przeanalizuj atrybuty klas pod kątem widoczności (modyfikatory public, private, etc.). Zastosuj możliwie najmniejszą widoczność.
10. Wygeneruj getter dla pola total w klasie DefaultCountingOutRhymer (opcja Source → Generate Getters and Setters).
11. Dokonaj hermetyzacji nieprywatnych atrybutów (polecenie Refactor → Encapsulate Field, w eclipse użyte z opcją keep field reference, w IntelliJ opcja Use accesors even when field is accessible powinna być odznaczona).
12. Usuń nieużywane setter (np. w Eclipse do analizy wywołań możesz użyć opcji Navigate → Open Call Hierarchy (ctrl+alt+h)).

- Po zakończeniu pracy nad tematem Accessory i hermetyzacja samodzielnie scal zmiany do gałęzi master przy użyciu mechanizmu Pull Request na platformie github.

Dział III: Testy, diagramy, dokumentacja + *gwiazdki (dla chętnych)

- Utwórz nową gałąź o nazwie **docs** i wykonaj na niej następujące działania:

13. Uporządkuj aplikację demo RhymersDemo (znajdującą się w *src/test/java*) rozbijając metodę *main*. Utwórz metodę statyczną testRhymers przyjmującą jako argument fabrykę stosów. W tym celu użyj opcji *Refactor* → *Extract Method* na zawartości funkcji *main* z pominięciem deklaracji lokalnej zmiennej *factory*.
14. *Znajdź klasę, która nie musi być publiczna. Przeorganizuj projekt tak, aby usunąć plik zawierający tę klasę.
15. Dokonaj walidacji projektu testami jednostkowymi. W razie potrzeby popraw testy i projekt. (Jeżeli nie było potrzeby poprawy to dodaj komentarz w kodzie, że projekt jest poprawny i zrób komit z tym komenatrzem)
16. *Napisz testy jednostkowe dla wybranej klasy projektu.
17. Wygeneruj automatycznie szkielet dokumentacji do wybranej klasy (dokumentacja ma być w pliku *java* klasy) i jej metod (np. opcja *Source* → *Generate Element Comment*) oraz ją uzupełnij.
18. *Przejrzyj implementację pod kątem jakości. Jeżeli znajdziesz jakieś potencjalne miejsca, które można naprawić, dodaj notkę „TODO:” w komentarzach, np.: `// TODO: needs refactoring - method shouldn't have 300 lines :)`

- Wprowadź zmiany z gałęzi **docs** do gałęzi master (na GitHub).
- Zgłoś rozwiązanie zadań z punktów 1-18 zgodnie z instrukcją akcji:

    - Na github-ie wyświetl stan repozytorium w gałęzi master z ostatniego commitu wykonanego przed rozpoczęciem pierwszych zajęć z tematu Git i Github (ostatni commit autorstwa prowadzących zajęcia). Użyj ikonki <> przy nazwie commitu.
    - W widoku w/w stanu repozytorium utwórz nową gałąź o nazwie **review**. Gałąź nie powinna zawierać commitów Twojego autorstwa.
    - Z gałęzi **master** zrób Pull Requesta do gałęzi **review**.
    - Dodaj prowadzącego jako reviewer'a Pull Requesta.