Test jednostkowy – testuje pojedynczy element programu. Metodę lub klasę – powinien testować tylko jedną funkcjonalność. Sprawdzamy czy dla podanych danych wejściowych, otrzymujemy przewidziane dane wyjściowe czy też zaplanowany wyjątek.

Testy pozwalają też na udokumnetowanie kodu lub sprawdzenie czy zewnętrzna biblioteka dalej działa w przewidywany sposób(przykładowo po aktualizacji).

Podejście TDD(Test Driven Development).
-Przed zaimplementowaniem funkcjonalności najpierw piszemy test sprawdzający jej poprawne działanie.

Składa się z 3 faz:
-red
-green
-refactor

Red, green nawiązuje do koloru w jakim z reguły IDE okresla czy test działa.
Po każdej rekatoryzacji odpalamy testy, aby sprawdzić czy nie pojawiły się błędy w kodzie.

W celu łatwego testowania funkcje powinny posiadać maksymalnie 2 argumenty, ewentualnie dopuszczalne są też 3.

Jedną z konwencji nazywania testów jest pisanie ich na zasadzie shouldDoWhenProvided
np. shouldThrowIllegalStateExceptionWhenPropertyNull.

Po co?
-testy jednostkowe wymuszają stosowanie się do zasady *Single Responsibility*
-pozwala to na łatwiejsze utrzymanie kodu
-pozwala na zmniejszenie liczby potencjalnych błędów i ich wcześniejsze wykrycie

inne narzędzia:
hamcrest – pozwala na pisanie testów jednostkowych w formie, która bardziej przypomina język ludzki. Nie ma problemu z rozróżnieniem, który argument funkcji *assertEquals* jest wartością oczekiwaną, a która testowaną.
Przykładowa asercja wartości wygląda w następujący sposób:

```
assertThat(outputSideDevice.getType(), is("output"));
assertThat(inputSideDeviceIds, hasSize(2));
assertThat(inputSideDeviceIds, containsInAnyOrder(4L, 5L));
```

mockito – pozwala zamockować zależności, czyli przykładowo zasymulować zachowanie funkcji. Przydatne gdy chcemy uniezależnić test od innej, nieznanej klasy – przykładowo łączącej się z bazą danych. Ponadto pozwala między innymi sprawdzenie czy i ilukrotnie została wywoałana jakaś metoda, przykład: *Mockito.verify(studentRepositoryMock).getAllStudents()*

Zamiast mockowania zależności, możemy też napisać **stub** czyli naszą implementację jakieś funkcjonalności na potrzeby testu. Przyklad *StudentServiceStubTest* wykorzystujący *StudentRepositoryStub*. Tym sposobem testujemy jedynie działanie metody w *StudentServiceStubTest*, a działanie funkcji ze *StudentRepository* jest dla nas przewidywalne. Wadą takiego rozwiązania jest to, że w przypadku poszerzenia interfejsu o nowe metody będzie konieczna ich implementacja w stubie oraz to, że przy pisaniu różnych scenariuszy testowych może pojawić się konieczność wielokrotnego powielania kodu dla tworzonych stubów.

Junit5 jest popularną biblioteką do pisania takich testów

Do pisania testów wykorzystuje się m.in. te adnotacje
@Test – oznacza testy
@BeforeEach – wykonuje się przed każdym testem
@AfterEach – wykonuje się po każdym teście
@BeforeAll – wykonuje się raz przed wszystkimi testami(funkcja musi być statyczna – uwaga! Niestatyczna klasa wewnętrzna nie może posiadać statycznych elementów)
@AfterAll – wykonuje się raz po wszystkich testach
@Nested – pozwala na zgrupowanie testów w ramach wewnętrznej klasy (przykład w GradeBookTest)
@Disabled – pomija test (wykorzystano w teście *shouldThrowIllegalStateExceptionWhenAddingIncorrectGrade)*

Przykładowe asercje
assertEquals – sprawdza czy wartość oczekiwana jest równa rzeczywistej
assertThrows – sprawdza czy rzucany jest podany wyjątek
assertAll – pozwala sprawdzić kilka asercji jednocześnie, nawet jeśli któraś się nie powiedzie, wykonane będą wszystkie

skróty klawiszowe dla Netbeans
alt + f6 - odpalenie testów dla projektu
ctrl + f6 – odpalenie pliku z testami
alt + shift + r – rezultaty tetstów

zbudowanie projektu *mvn install*
z pominięciem testów `mvn install -DskipTests`

uruchomienie testów *mvn test*

W przypadku gdy nie ma maven

wget http://ftp.man.poznan.pl/apache/maven/maven-3/3.6.1/binaries/apache-maven-3.6.1-bin.tar.gz
tar -xvf apache-maven-3.6.1-bin.tar.gz
 vim ~/.bashrc

PATH=$PATH:/home/stud2016/*6nazwisko*/maven/apache-maven-3.6.1/bin
M2_HOME="/home/stud2016/*6nazwisko*/maven/apache-maven-3.6.1"