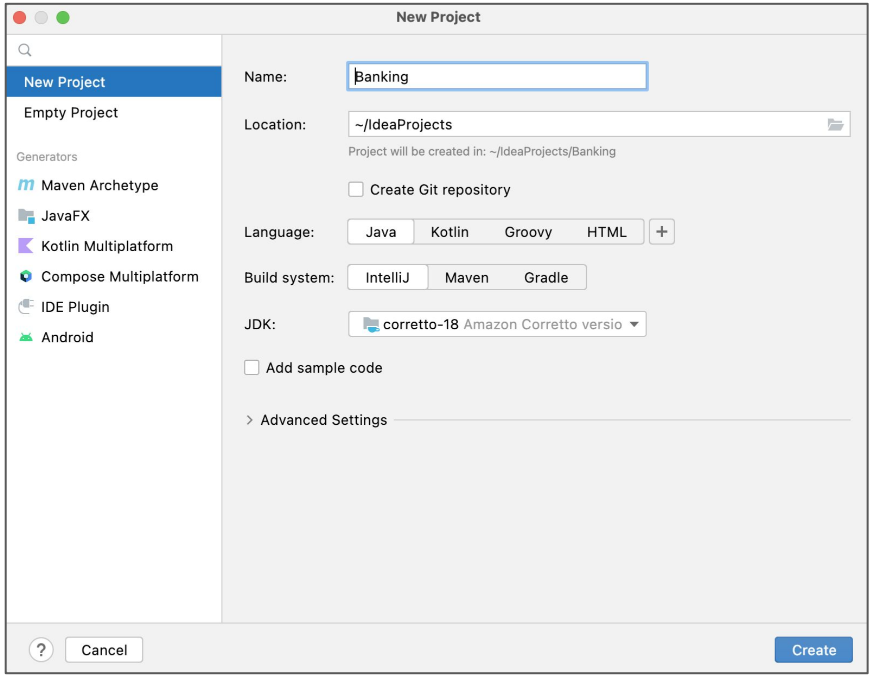
CSCI 2120:
Software Design & Development II

Testing, Unit Testing & JUnit

JUnit + IntelliJ

Demo: JUnit testing for a Banking Project

- Make new Java project in IntelliJ named Banking
- Create a new Java class named BankAccount



New Project

Name: Banking

Location: ~/IdeaProjects

Project will be created in: ~/IdeaProjects/Banking

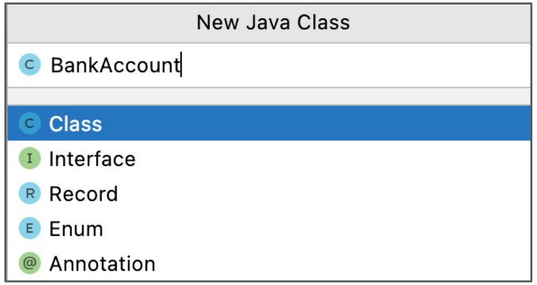
Create Git repository

Language: Java | Kotlin | Groovy | HTML

Build system: IntelliJ | Maven | Gradle

JDK: corretto-18 Amazon Corretto versio

Add sample code



New Java Class

BankAccount

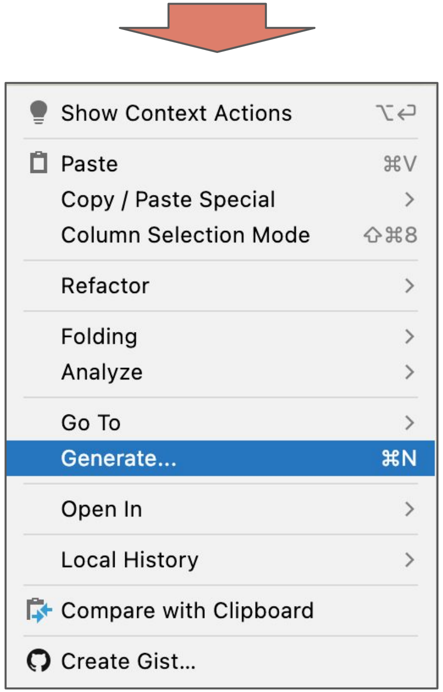
class BankAccount

```java
public class BankAccount {

   private String firstName;
   private String lastName;
   private double balance;

   public BankAccount(String firstName, String lastName, double balance) {
       this.firstName = firstName;
       this.lastName = lastName;
       this.balance = balance;
   }

   public double deposit(double amount, boolean branch){
       balance += amount;
       return balance;
   }

   public double withdraw(double amount, boolean branch){
       balance -= amount;
       return balance;
   }

   public double getBalance() {
       return this.balance;
   }
}
```

JUnit: Use IntelliJ to add JUnit into the Banking project

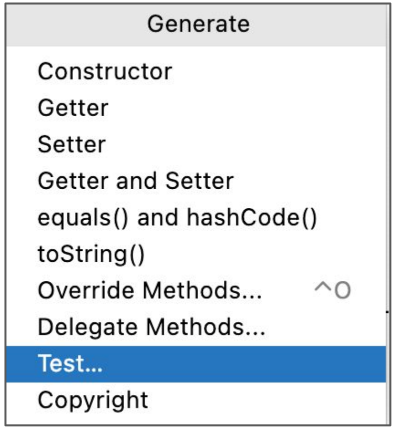
Right click on the class name **`BankAccount`** to open menu options

1



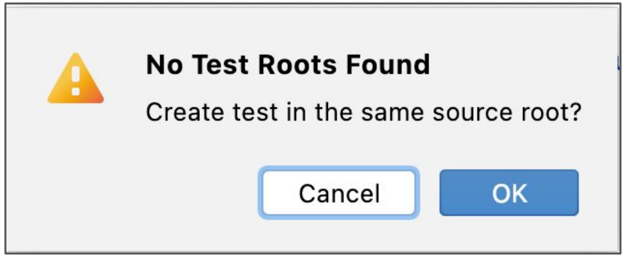
Select 'Generate' option

2



Select 'Test…' option

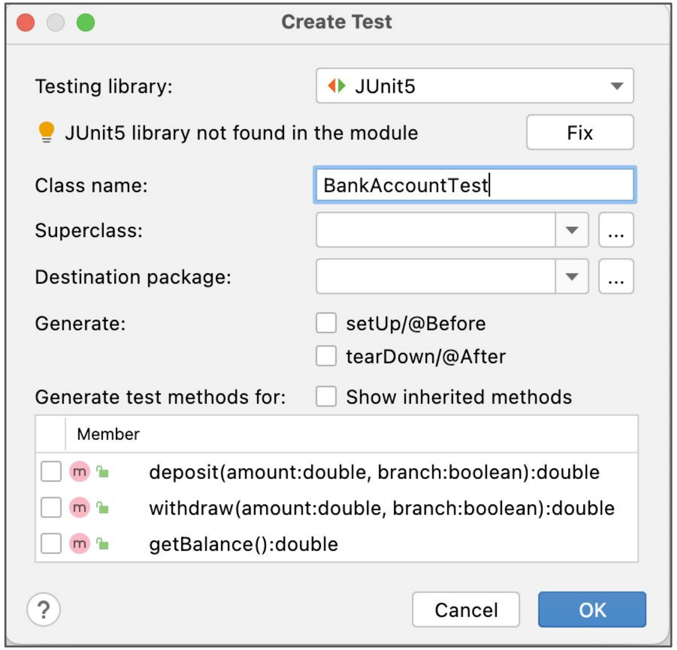
3



Select 'OK' option

JUnit: Use IntelliJ to add JUnit into the Banking project

4


Create Test

Testing library: JUnit5

JUnit5 library not found in the module

Class name: BankAccountTest

Superclass:

Destination package:

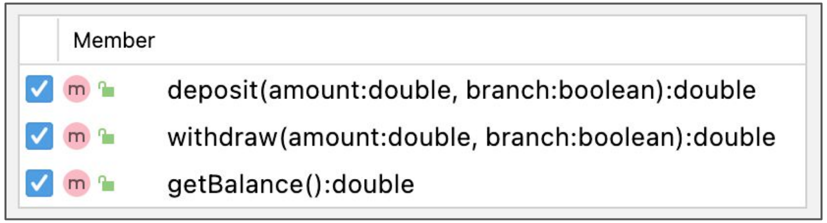
Generate: setUp/@Before [ ], tearDown/@After [ ]

Generate test methods for: Show inherited methods [ ]

	Member
☐	deposit(amount:double, branch:boolean):double
☐	withdraw(amount:double, branch:boolean):double
☐	getBalance():double

- Select 'JUnit5' as Testing library option
- Use 'BankAccountTest' as Class name
- Perform Step 5
- Finally, Select 'OK'

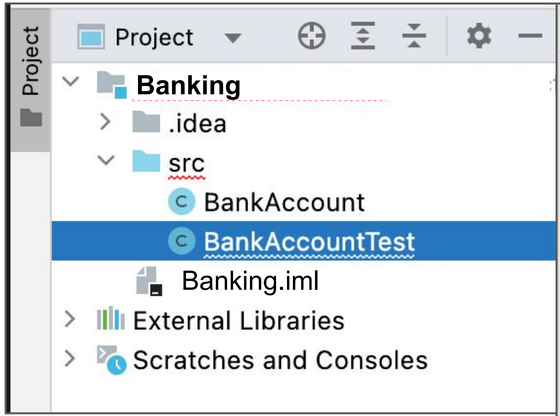
5

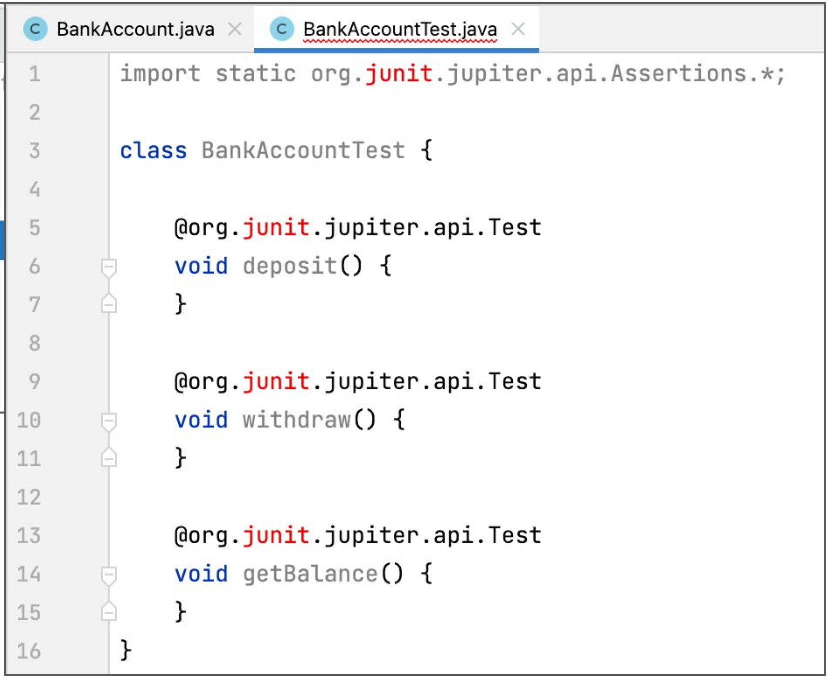


	Member
✓	deposit(amount:double, branch:boolean):double
✓	withdraw(amount:double, branch:boolean):double
✓	getBalance():double

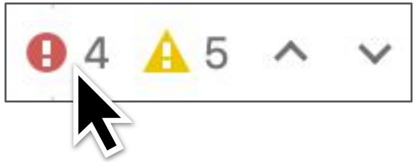
- Select all instance methods as JUnit test methods

BankAccountTest: Result from the `Test...` setup





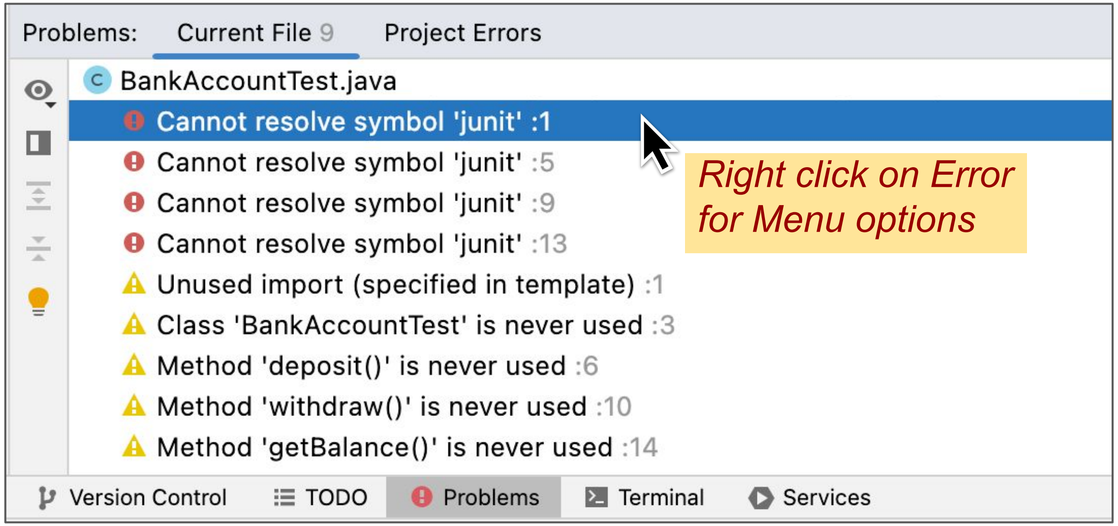
```java
import static org.junit.jupiter.api.Assertions.*;

class BankAccountTest {

    @org.junit.jupiter.api.Test
    void deposit() {
    }

    @org.junit.jupiter.api.Test
    void withdraw() {
    }

    @org.junit.jupiter.api.Test
    void getBalance() {
    }
}
```



Select to see the warning & error log

The red text & squiggly lines highlights that there are errors in the source code

Error log: Resolving problems



Problems: Current File 9 Project Errors

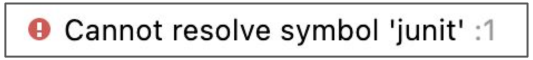
BankAccountTest.java

```
Cannot resolve symbol 'junit' :1
Cannot resolve symbol 'junit' :5
Cannot resolve symbol 'junit' :9
Cannot resolve symbol 'junit' :13
Unused import (specified in template) :1
Class 'BankAccountTest' is never used :3
Method 'deposit()' is never used :6
Method 'withdraw()' is never used :10
Method 'getBalance()' is never used :14
```

Right click on Error for Menu options

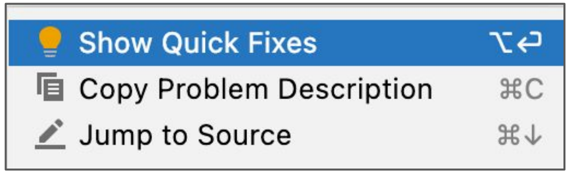
Problem: JUnit is not in the project's *External Libraries*

1

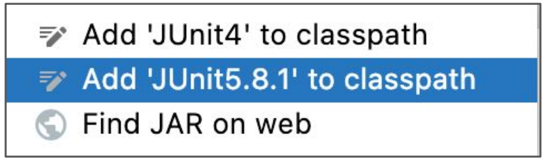

Cannot resolve symbol 'junit' :1

Right click on Error for Menu options

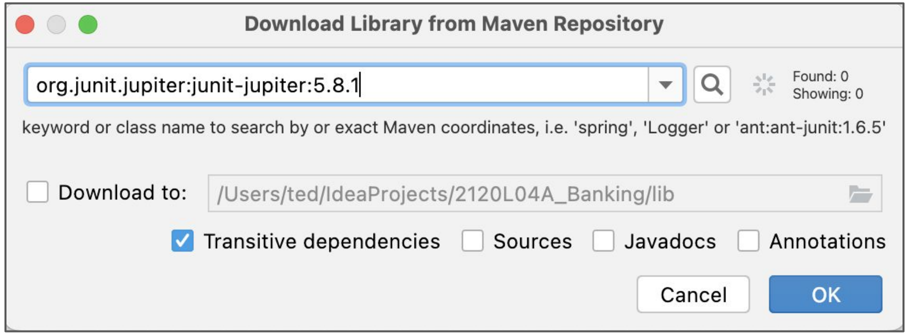
2



3



4


Download Library from Maven Repository

org.junit.jupiter:junit-jupiter:5.8.1

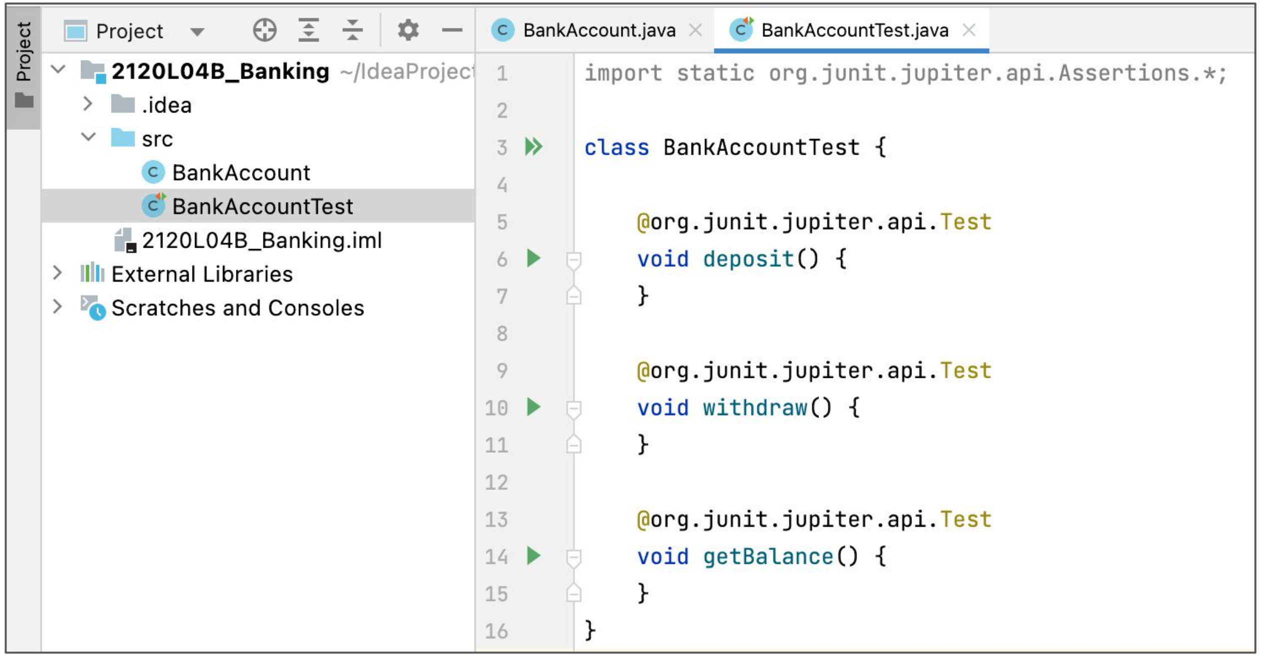
Found: 0
Showing: 0

keyword or class name to search by or exact Maven coordinates, i.e. 'spring', 'Logger' or 'ant:ant-junit:1.6.5'

☐ Download to: /Users/ted/IdeaProjects/2120L04A_Banking/lib

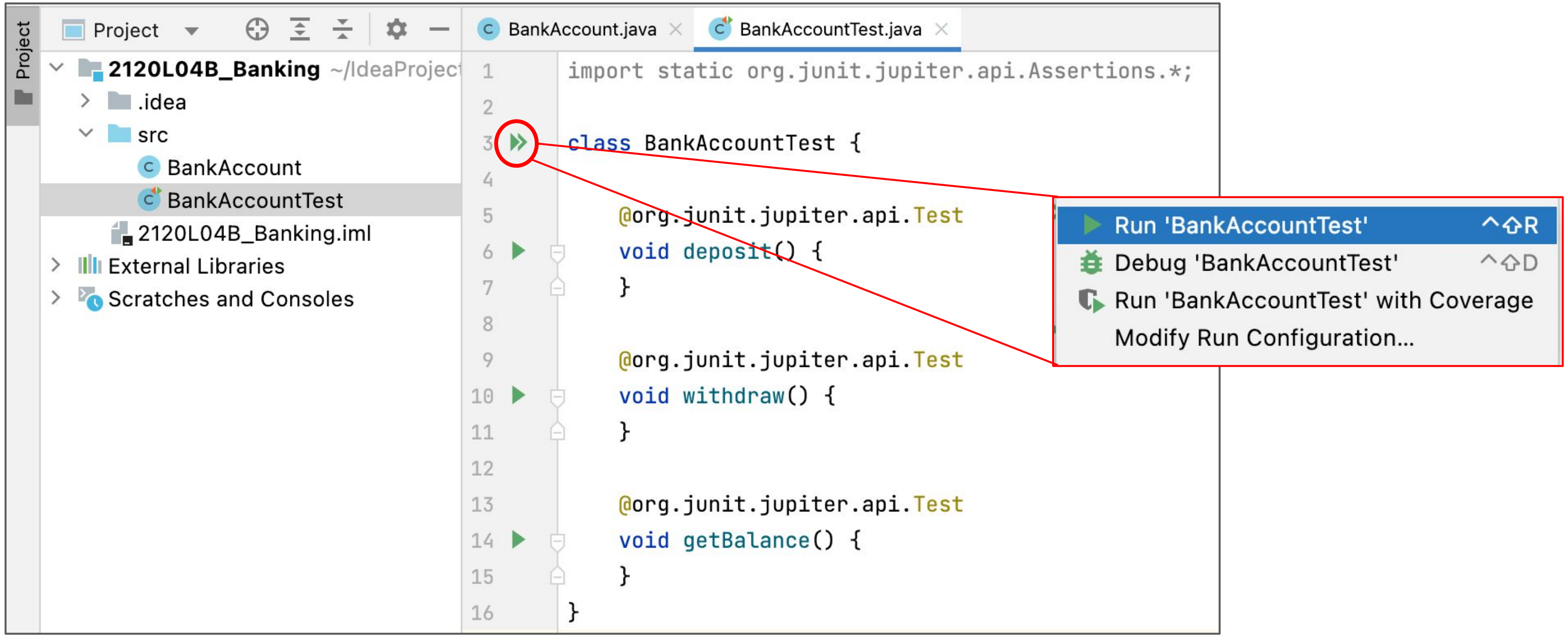
✓ Transitive dependencies ☐ Sources ☐ Javadocs ☐ Annotations

Problem: Resolved - *i.e. no red text.*



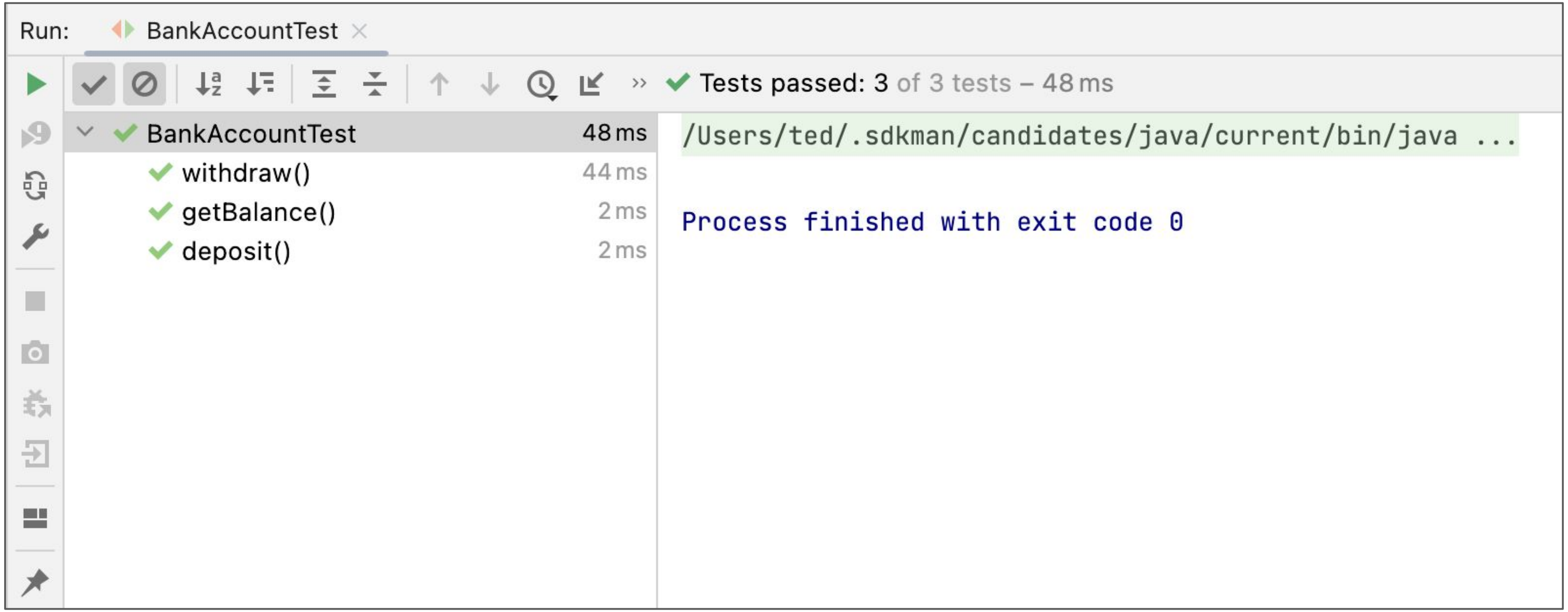
```
import static org.junit.jupiter.api.Assertions.*;

class BankAccountTest {

    @org.junit.jupiter.api.Test
    void deposit() {
    }

    @org.junit.jupiter.api.Test
    void withdraw() {
    }

    @org.junit.jupiter.api.Test
    void getBalance() {
    }
}
```

Test: Click on ⏩ `BankAccountTest` to run JUnit


Project
2120L04B_Banking ~/IdeaProject
.idea
src
BankAccount
BankAccountTest
2120L04B_Banking.iml
External Libraries
Scratches and Consoles
BankAccount.java
BankAccountTest.java
Run 'BankAccountTest' ^⇧R
Debug 'BankAccountTest' ^⇧D
Run 'BankAccountTest' with Coverage
Modify Run Configuration...

```
import static org.junit.jupiter.api.Assertions.*;

class BankAccountTest {

    @org.junit.jupiter.api.Test
    void deposit() {
    }

    @org.junit.jupiter.api.Test
    void withdraw() {
    }

    @org.junit.jupiter.api.Test
    void getBalance() {
    }
}
```

Test: It works, but they all Pass by default.



✓ Tests passed: 3 of 3 tests – 48 ms

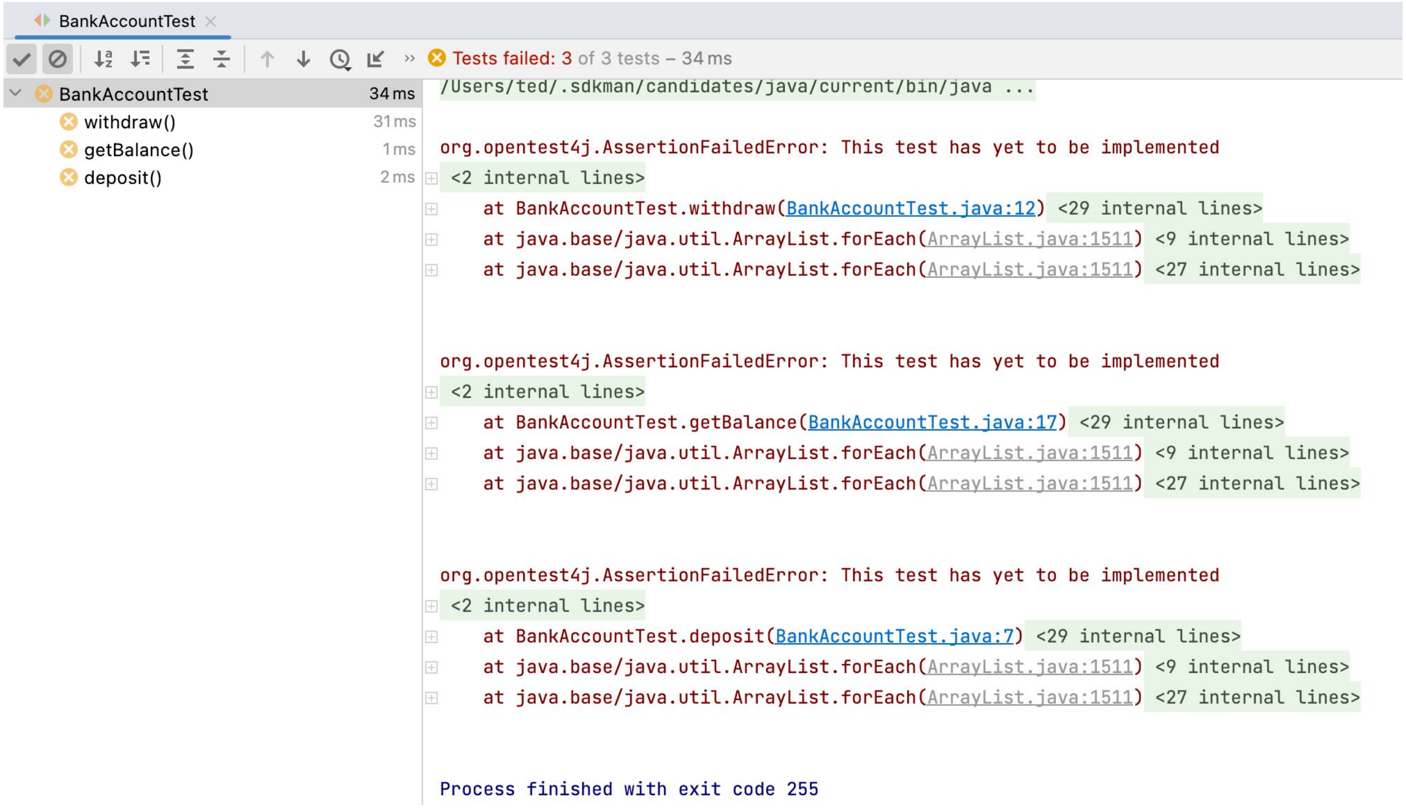
```
BankAccountTest   48 ms
  withdraw()      44 ms
  getBalance()     2 ms
  deposit()        2 ms
```

```
/Users/ted/.sdkman/candidates/java/current/bin/java ...


Process finished with exit code 0
```

BankAccountTest: method fail() forces a fail result

```
import static org.junit.jupiter.api.Assertions.*;

class BankAccountTest {

   @org.junit.jupiter.api.Test
   void deposit() {
       fail("This test has yet to be implemented");
   }

   @org.junit.jupiter.api.Test
   void withdraw() {
       fail("This test has yet to be implemented");
   }

   @org.junit.jupiter.api.Test
   void getBalance() {
       fail("This test has yet to be implemented");
   }
}
```

Test: Run JUnit ⏩, where all test should Fail


BankAccountTest

Tests failed: 3 of 3 tests – 34 ms

```
BankAccountTest   34 ms
  withdraw()      31 ms
  getBalance()     1 ms
  deposit()        2 ms
```

```
/Users/ted/.sdkman/candidates/java/current/bin/java ...

org.opentest4j.AssertionFailedError: This test has yet to be implemented
 <2 internal lines>
    at BankAccountTest.withdraw(BankAccountTest.java:12) <29 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <27 internal lines>


org.opentest4j.AssertionFailedError: This test has yet to be implemented
 <2 internal lines>
    at BankAccountTest.getBalance(BankAccountTest.java:17) <29 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <27 internal lines>


org.opentest4j.AssertionFailedError: This test has yet to be implemented
 <2 internal lines>
    at BankAccountTest.deposit(BankAccountTest.java:7) <29 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <27 internal lines>


Process finished with exit code 255
```

`BankAccountTest`: dummyTest() & assertEquals

```
@org.junit.jupiter.api.Test
void dummyTest() {
   assertEquals(1,2);
}
```

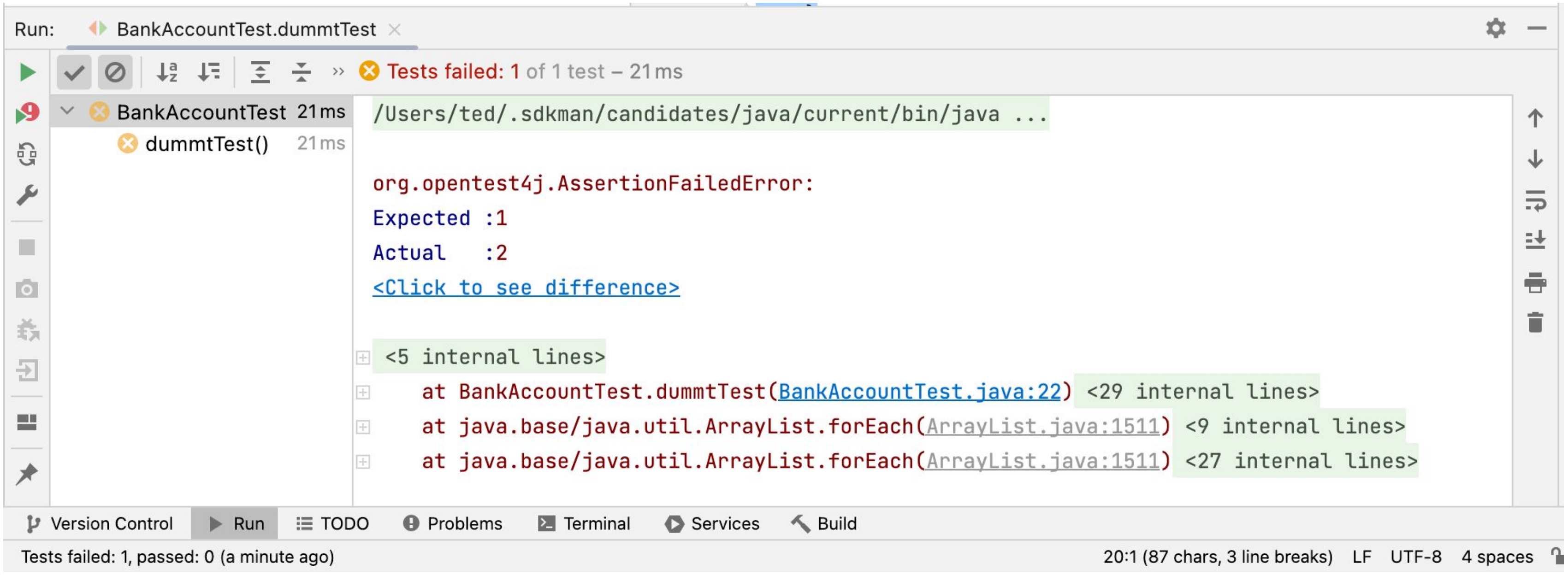
- Add a @Test method named dummyTest to the class BankAccountTest,
- Use the `assertEquals` method, hardcode two different values as params to try it out!

JUnit JavaDocs

`assertEquals( expected, actual )`	Assert that expected and actual values are equal.

API: https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html

Test: Run JUnit ▶ on dummyTest() method



```
Tests failed: 1 of 1 test – 21ms

/Users/ted/.sdkman/candidates/java/current/bin/java ...

org.opentest4j.AssertionFailedError:
Expected :1
Actual   :2
<Click to see difference>

<5 internal lines>
    at BankAccountTest.dummtTest(BankAccountTest.java:22) <29 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <27 internal lines>
```

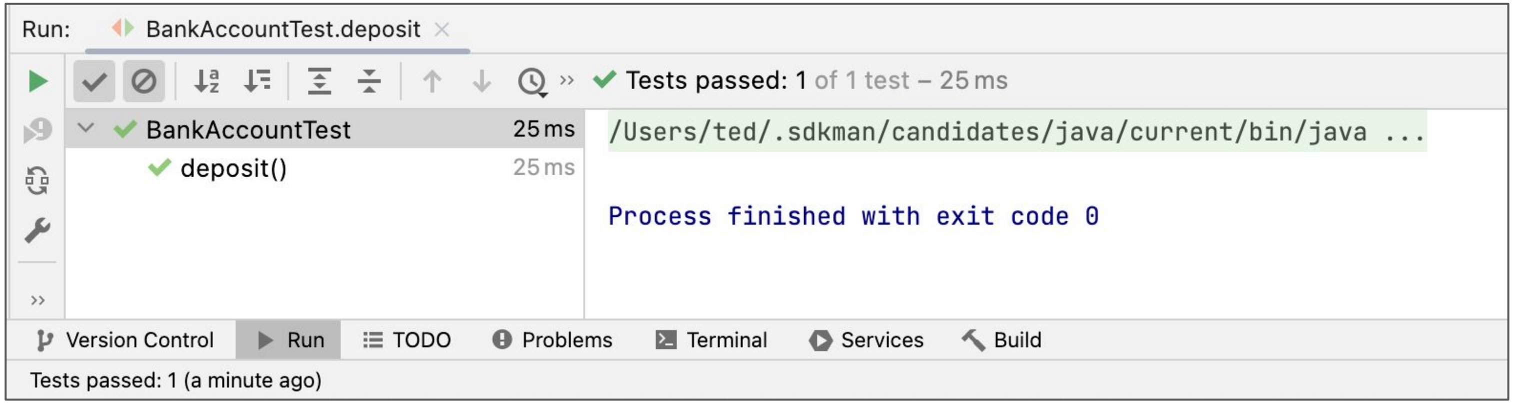
Tests failed: 1, passed: 0 (a minute ago) 20:1 (87 chars, 3 line breaks) LF UTF-8 4 spaces

- JUnit test fails and logs both the expected and actual values.

BankAccountTest: @Test method: deposit()

```
@org.junit.jupiter.api.Test
void deposit() {
   BankAccount account = new BankAccount("Ted", "Holmberg", 100.00);
   double balance = account.deposit(75.00, true);
   assertEquals(175.00, balance, 0);
}
```

Test: Run JUnit ▶ on @Test method deposit()



Tests passed: 1 of 1 test – 25 ms

```
BankAccountTest   25 ms
  deposit()       25 ms
```

```
/Users/ted/.sdkman/candidates/java/current/bin/java ...

Process finished with exit code 0
```

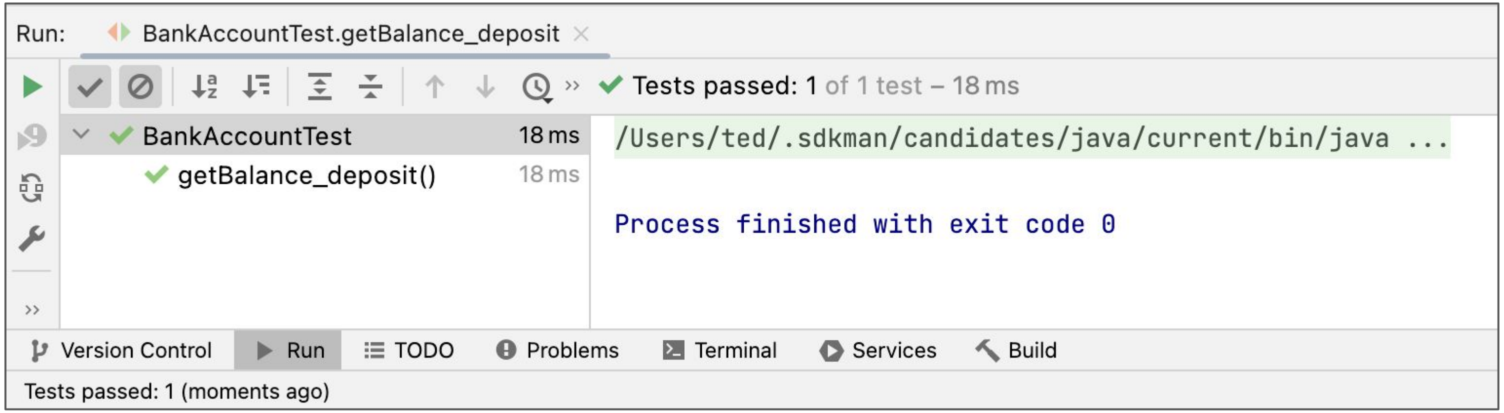
Tests passed: 1 (a minute ago)

BankAccountTest: @Test method: getBalance_deposit()

```
@org.junit.jupiter.api.Test
void getBalance_deposit() {
   BankAccount account = new BankAccount("Ted", "Holmberg", 100.00);
   account.deposit(75.00, true);
   assertEquals(175.00, account.getBalance(), 0);
}
```

*Unit testing convention is to apply a setter then check with getter

Test: Run JUnit ▶ on @Test method getBalance_deposit()



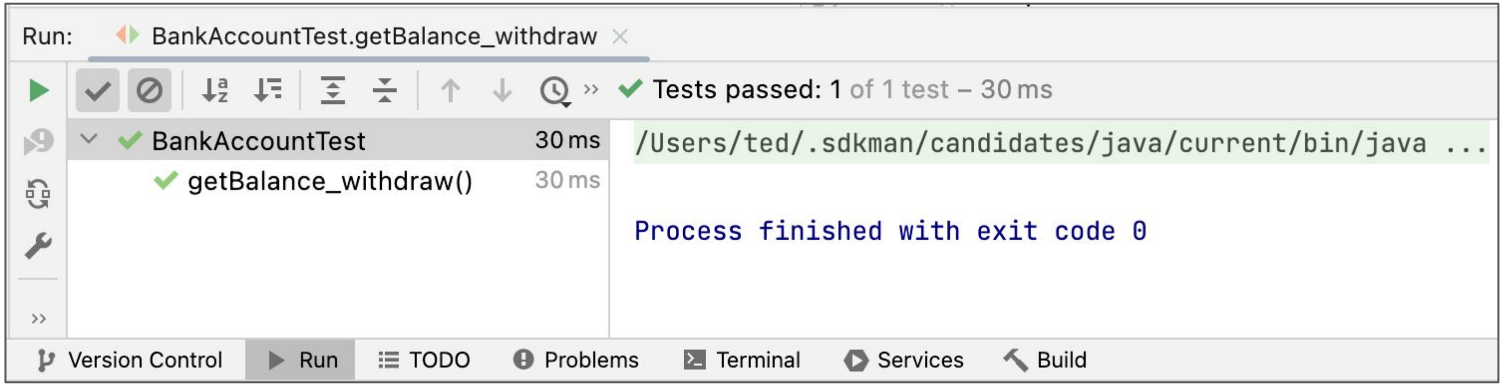
```
Tests passed: 1 of 1 test – 18 ms

BankAccountTest          18 ms
   getBalance_deposit()  18 ms

/Users/ted/.sdkman/candidates/java/current/bin/java ...

Process finished with exit code 0

Tests passed: 1 (moments ago)
```

BankAccountTest: @Test method: getBalance_withdraw()

```
@org.junit.jupiter.api.Test
void getBalance_withdraw() {
   BankAccount account = new BankAccount("Ted", "Holmberg", 100.00);
   account.withdraw(75.00, true);
   assertEquals(25.00, account.getBalance(), 0);
}
```

*Unit testing convention is to apply a setter then check with getter

Test: Run JUnit ▶ on @Test method getBalance_withdraw()



```
✓ Tests passed: 1 of 1 test – 30 ms

✓ BankAccountTest            30 ms
   ✓ getBalance_withdraw()   30 ms

/Users/ted/.sdkman/candidates/java/current/bin/java ...

Process finished with exit code 0
```

BankAccount: Refactor code

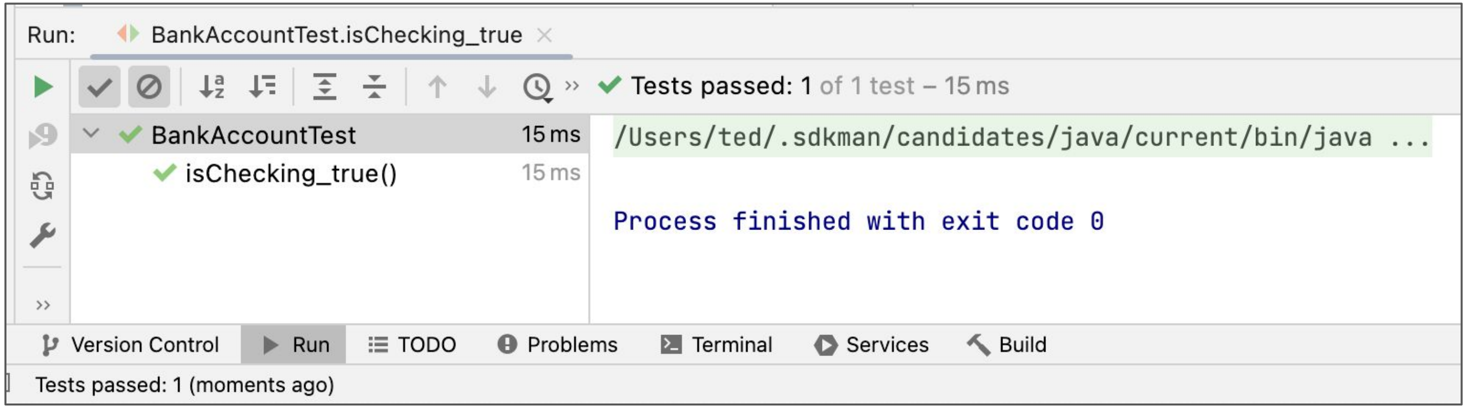
```
public class BankAccount {
   private String firstName;
   private String lastName;
   private double balance;

   public static final int CHECKING = 1;
   public static final int SAVINGS = 2;
   private int accountType;

   public BankAccount(String firstName, String lastName, double balance, int accountType) {
       this.firstName = firstName;
       this.lastName = lastName;
       this.balance = balance;
       this.accountType = accountType;
   }

   public boolean isChecking() {
       return accountType == CHECKING;
   }

...
```

BankAccountTest: @Test method: isChecking_true()

```
@org.junit.jupiter.api.Test
void isChecking_true() {
   BankAccount account = new BankAccount("Ted", "Holmberg", 100.00, BankAccount.CHECKING);
   assertTrue(account.isChecking(), "The account is not checking");
}
```

*assertTrue() has two parameters, 1: boolean value, 2: text message on failure

Test: Run JUnit ▶ on @Test method isChecking_true()



```
✓ Tests passed: 1 of 1 test – 15 ms
✓ BankAccountTest        15 ms
   ✓ isChecking_true()   15 ms

/Users/ted/.sdkman/candidates/java/current/bin/java ...

Process finished with exit code 0

Tests passed: 1 (moments ago)
```

JUnit: BeforeAll vs BeforeEach

- If the tests don't make any changes to those conditions, you can use beforeAll (which will run once).

- If the tests *do* make changes to those conditions, then you would need to use beforeEach, which will run before every test, so it can reset the conditions for the next one.

It is safest to default to using beforeEach as it reduces the opportunity for human error, i.e. not realizing that one test is changing the setup for the next one.

BankAccountTest: Refactor code

```
class BankAccountTest {
    private BankAccount account;

    @org.junit.jupiter.api.BeforeEach
    public void setup(){
        account = new BankAccount("Ted", "Holmberg", 100.00, BankAccount.CHECKING);
        System.out.println("Running a test...");
    }

    @org.junit.jupiter.api.Test
    void deposit() {
        double balance = account.deposit(75.00, true);
        assertEquals(175.00, balance, 0);
    }
    @org.junit.jupiter.api.Test
    void getBalance_deposit() {
        account.deposit(75.00, true);
        assertEquals(175.00, account.getBalance(), 0);
    }
    @org.junit.jupiter.api.Test
    void getBalance_withdraw() {
        account.withdraw(75.00, true);
        assertEquals(25.00, account.getBalance(), 0);
    }
    @org.junit.jupiter.api.Test
    void isChecking_true() {
        assertTrue(account.isChecking(), "The account is not checking");
    }
}
```

BankAccount: Refactor method `withdraw()`

```
public double withdraw(double amount, boolean branch){
   if ((amount > 500.00) && !branch){
       throw new IllegalArgumentException("Cannot Withdraw");
   }
   balance -= amount;
   return balance;
}
```

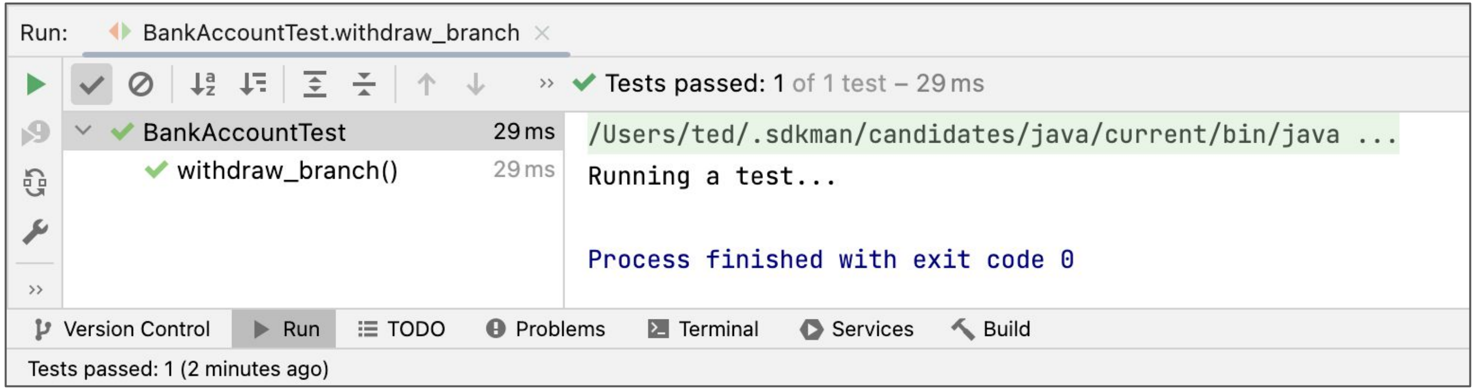
*Method that throws an Exception, so that we can test for it in JUnit

`BankAccountTest`: @Test method: `withdraw_branch()`

```java
@org.junit.jupiter.api.Test
void withdraw_branch() throws  Exception{
   double balance = account.withdraw(60.00, true);
   assertEquals(40.00, balance, 0);
}
```

*assertEquals(double, double) has three parameters, 1: expected, 2: actual, 3: delta

Test: Run JUnit ▶ on @Test method `withdraw_branch()`



✓ Tests passed: 1 of 1 test – 29 ms

✓ BankAccountTest 29 ms
✓ withdraw_branch() 29 ms

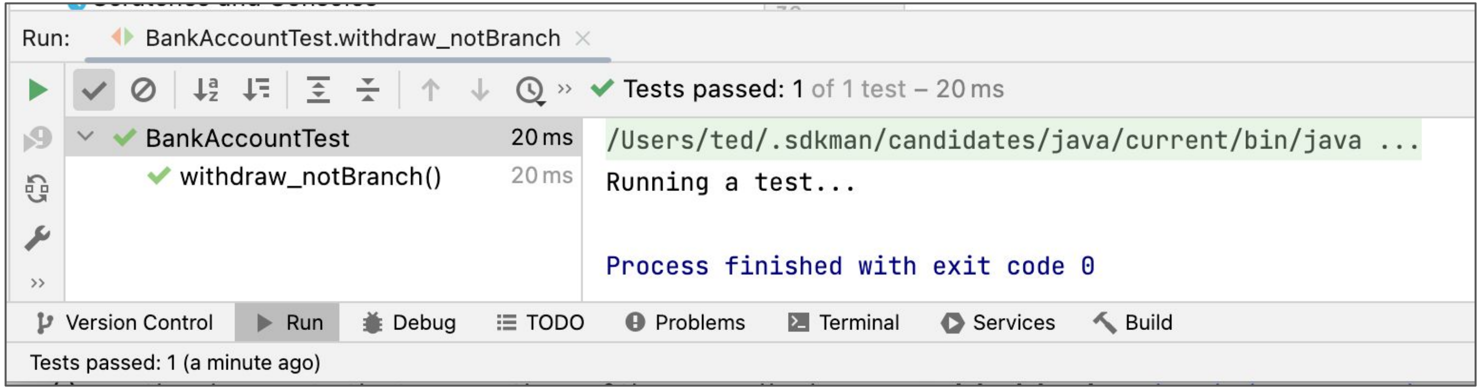
```
/Users/ted/.sdkman/candidates/java/current/bin/java ...
Running a test...

Process finished with exit code 0
```

Tests passed: 1 (2 minutes ago)

BankAccountTest: @Test method: withdraw_notBranch()

```
@org.junit.jupiter.api.Test
void withdraw_notBranch() throws Exception{
   IllegalArgumentException thrown = assertThrows(IllegalArgumentException.class, () -> {
       //Code under test
       double balance = account.withdraw(600.00, false);
   });
   assertEquals("Cannot Withdraw", thrown.getMessage());
}
```

Test: Run JUnit ▶ on @Test method withdraw_notBranch()


Run: BankAccountTest.withdraw_notBranch

Tests passed: 1 of 1 test – 20 ms

BankAccountTest 20 ms
withdraw_notBranch() 20 ms

```
/Users/ted/.sdkman/candidates/java/current/bin/java ...
Running a test...

Process finished with exit code 0
```

Tests passed: 1 (a minute ago)

Demo Design Strategy: Implementation-Driven Testing

In this demo, we implemented the project code first and then unit tested it. Such an approach is called **Implementation-Driven Testing** or (Implementation Testing).

Alternative Strategy: Test-Driven Development (TDD)

- Test-Driven Development or *(Test-Driven Implementation)* is a better approach for designing complex software systems reverses the order. You would create the JUnit test classes first and then build your project code against the test fixture and test cases.

- Such an approach allows every developer on the team to use the test classes as a means to mock interactions within the system. So long as your classes pass all the test, they are ready to be integrated into the software system.

- This approach allows multiple developers to work concurrently on code that might normally be interdependent on each other.

END