Starter Code

The Github Classroom repository for the exam includes starter code for both problems, as follows:

`README.md`	– Template for README. Please expand this. Write descriptive README to show that you care about your work.
`doc/`	– Javadoc for the Auction classes.
`test/auction/AuctionTest.java`	- example unit tests.
`auction.jar`	- JAR file containing auction.Auction and auction.AuctionError Add this JAR file to your project (each IDE has a different way to do this).

Problem: Unit Tests for Auction class

Write unit tests for the Auction class, which is contained in the auction.jar file.

The Auction class contains 7 variants, numbered 1 - 7. One of them is correct (I hope) and the others contain at least one error according to the specification of what the auction *should do*.

You must write unit tests to identify each of the defects and create a table in your project README.md what contains information similar to this:

AuctionType	Failing Test	Description
1	testBiddingWhenAuctionStopped	Bids are accepted when auction is stopped.
2	testBidTooLow	Auction accepts a new bid lower than the current highest bid (it should throw exception).

Use your unit test names in the table.

How the Auction Works

An Auction is for accepting bids on some thing.

An auction accepts bids from participants, who place bids by invoking bid("*bidder name*", *amount*). Each new bid amount must be higher than the highest bid so far, and the *bidder name* may not be blank or null. Each Auction has a minimum bidding *increment* (the default is 1.0). A new bid must exceed the highest bid so far by at least this increment. The minium increment can also be given as a parameter to the constructor:

auction = new Auction("Samsung S10", 100) ;

- create an auction for a Samsung S10 with minimum bidding increment of **100**.

The minimum bidding increment is 100, so if the best bid so far is 1,000 then someone can bid 1100 or 1101 or 1250 but not 1099.

auction.start() enables bidding.

auction.stop() disables bidding.

auction.isEnabled() test if bidding is allowed.

auction.bid("Name", amount) submits a bid by "Name". The amount bid must be greater than the current highest bid by at least the auction bidding increment. If a bid is too low or the auction is stopped, then an AuctionError exception is thrown.

If the bid amount is non-postive (bid<=0) than IllegalArgumentException is thrown, and if bidder name is null or contains only space then IllegalArgumentException is thrown.

auction.bestBid() returns the best bid so far and auction.winner() returns name of person who submitted the best bid. For a new Auction, bestBid() is 0 and winner() is "no bids".

auction.winner() - returns the name of person who submitted the best bid.

AuctionError:

Bids (calls to auction.bid(name,amount)) are allowed only while an bidding is enabled. To enable bidding, call auction.start() and to disable bidding call auction.stop(). You can start - stop bidding many times. If auction.bid() is called when the auction is stopped, the Auction throws an AuctionError.

If a bid is too low (but a positive number) it also throws AuctionError.

Auction Rules

1. Each bid must have an amount > 0 and a non-blank, non-null bidder name (string). If this rule is violated, an IllegalArgumentException is thrown.

2. Bid amount must be at least the current best bid plus a *minimum increment*.
The minimum increment is specified as parameter in the Auction constructor (default increment is 1). If a bid is too low (but > 0) then an AuctionError is thrown.

3. Bids are allowed only when the auction is active. Auction has methods start() and stop() to enable and disable bidding. start/stop may be called many times to pause the auction. A new Auction object is initially in the stopped state. If a bid is placed when the auction is stopped, an AuctionError is thrown.

4. The participant with the highest valid bid at any time is the (current) winner.

Assignment

Create a unit test class named **`AuctionTest.java`** containing unit tests for Auction.

1. Write unit tests to test that the Auction methods obey the rules of the auction and behavior in the Javadoc for the methods. Verify that the auction methods correctly implement the auction behavior; don't just test individual methods.

2. Create your own test cases, but you should test for (at least) this behavior:

- a new Auction is in the correct state, and has the expected initial values
- bidding is allowed only when an Auction is started, and the correct exception is thrown otherwise
- bidding is not allowed until a new auction starts accepting bids (auction.start()).
- bidding works correctly and the winner name and best bid are always correct (even if no bids), even if there is only one bid or one person bids many times.
  In particular, a new bid is accepted only if it exceeds the previous best bid by a minimum bidding increment. Test an auction with increment = 1 (default) and other values.
- the winner and best bid are correct even if some invalid bid is entered after the best bid is placed! This is a common form of error, where some invalid data corrupts the state of an object.
- Illegal bids handled correctly, such as low bids or bids when auction is stopped. The correct exception is always thrown.

3. Write tests based on the ***specification*** for the methods. See the Javadoc for details. Don't test based on the way the auction *actually behaves* (which may be buggy).

4. For tests that expect the code to throw an exception, be careful how you structure the tests! If you invoke 2 methods in one test, it is possible that one method doesn't throw exception (when it should) but the test will not detect this because the other method call always throws exception.

5. Write many test methods, where each one tests a specific thing. Each method can contain more than one assert. Don't write just one or two long test methods.

6. Use descriptive test method names such as testBiddingWhenAuctionIsStopped.

7. Use a setUp method to create an Auction object as a *test fixture* (an attribute) instead of creating an auction in each test. It is OK for some test(s) to create their own auction object if you need to change some parameters of the auction.

Simple test for best bid:

```
@Test
public void testAuctionWithOneBidder( ) {
    final double TOL = 1.0E-4;
    auction.start();
    auction.bid("Jittat", 500.0);
    assertEquals("Jittat", auction.winner() );
    assertEquals(500.0, auction.bestBid(), TOL);
    // can he bid again?
    auction.bid("Jittat", 520.0);
    assertEquals("Jittat", auction.winner() );
    assertEquals(520.0, auction.bestBid(), TOL);
}
```

For tests that expect an exception to be thrown, use:

```
@Test(expected=auction.AuctionError.class)
public void testBiddingNotAllowedWhenStopped() {
    auction.stop();
    auction.bid("Taksin", 1000000.0);  // should throw AuctionError
}
```

How to Set the Auction Type

The Auction class contains 7 different variants of the Auction code. You must test all 7 variants and find the defects.

There are 3 ways to set the auction type, listed here from highest precedence to lowest precedence.

1. Call **`auction.setAuctionType(int type)`** after creating an Auction object. Use **`type`** = 1 to 7. This is the easiest way for testing.

2. Set a system property. You can do this on the Java command line or by calling System.setProperty() *before* you create the Auction object:

```
java -DAUCTIONTYPE=3  auction.Auction
```

or (in your main method or JUnit setUp method):

```
System.setProperty("AUCTIONTYPE", "3");
```

3. Set an environment variable named AUCTIONTYPE. This must be done in the *same terminal or shell* that you run the Auction application.

Linux/MacOS: **`export AUCTIONTYPE=3`** (any value 1-7)

Windows/DOS: **`set AUCTIONTYPE=3`**

Writing Parameterized Tests

You need to run all the unit tests for each auction type 1, 2, ..., 7. That's repetitive.

JUnit has a feature call *parameterized tests* that lets you re-run a set of tests using multiple values for some parameters. The values of the parameters are *injected* into some public variables, using parameter data that you specify.

Don't do this until you have already become familiar with JUnit, because it is added complexity.
How to write parameterized tests: https://dzone.com/articles/junit-parameterized-test
Example parameterized test code will be posted on Google Classroom.

Example Auction
Please do not copy this example. Create your own test cases.

`auction = new Auction("Core Java")`	Create auction for a great Java book!
`auction.isEnabled()` `false`	New auction is stopped. Call start() to enable bidding.
`auction.bestBid()` `0.0`	No bids yet.

`auction.start( )`
`auction.isEnabled()`
`true`
	Start accepting bids.

`auction,bid("Jim", 200)`	Jim bids 200 Baht.
`auction.bid("mai", 250)`	Mai bids 250 Baht.
`auction.bid("", 1000)` `IllegalArgumentException: Invalid bidder name`	Anonymous bids are not allowed.

`auction.bid("Jim", 250.50)`
 	
`AuctionError: Bid is too low`
	What a cheap-stake! Only 50 *satang* higher!
	But the bid is not allowed. The default bidding increment is 1. New bid must be at least 251 Baht.

`auction.bestBid()`
`250`
`auction.winner( )`
`'Mai'`
	You can call these methods at any time to see the best bid so far and name of top bidder.
	Names are converted to Title Case and excess space removed.

`auction.bid("Jittat", 260)`	Ajarn Jittat wants the book, too!
`auction.stop()`	No bidding allowed now. Call start( ) again to re-enable bidding.
`auction.bid("Jim", 270)` `AuctionError: bidding not allowed now`	Too late! Auction has stopped.
`auction.winner()` `'Jittat'`	And he gets "*Core Java*" at a bargain price!