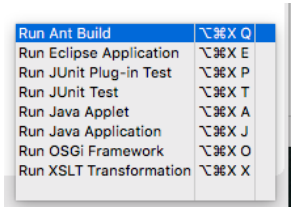
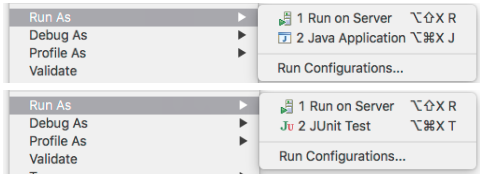
Useful Keyboard Shortcuts in Eclipse

- Run as a Java App
  - Alt + Cmd + X, (and then) J
- Run as a JUnit Test
  - Alt + Cmd + X, (and then) T
- Run as an Ant script
  - Alt + Cmd + X, (and then) Q





Just in case…

- Run
  - Cmd + Shift + F11
  - Runs the current (open) Java class if it has main().
    - If not, runs the last launched class.
  - Starts JUnit and run test cases if a test class is open.
- Debug
  - Cmd + F11
- Quick fix
  - Cmd + 1
- Code assist
  - Ctrl + Space
- Organize import statements
  - Ctrl + Shift + O

Useful Eclipse Plugins

- Quick JUnit
  - https://marketplace.eclipse.org/content/quick-junit
  - https://github.com/kompiro/quick-junit
  - Cmd/Ctrl + 9
    - Move from a tested class (XYZ) to its test class (XYZTest), and vice versa.
      - If a test class is not defined yet, pop up a wizard to do so.
  - Cmd/Ctrl + 0
    - Run a test class with JUnit
    - Easier to type/remember than Shift+Cmd+X → T
    - Run a test method with JUnit if a cursor is placed in the test method.
  - Cmd/Ctrl + Shift + 0
    - Run a test class with the debugger

- JUnit Helper
  - https://marketplace.eclipse.org/content/junit-helper
  - https://github.com/seratch/junithelper
  - ALT + 9
    - Generates a test class for a tested class based on a template.
      - static imports, a test class and test methods with template-based bodies

```
public class SavingsAccount{
 public SavingsAccount(
              float balance){...}
 public float computeTax(){...}
}
```

```
public void computeTax_(){
  float balance = 0.0F;
  SavingsAccount target =
       new SavingsAccount(balance);
  float actual = target.computeTax();
  float expected = 0.0F;
  assertThat(actual,
            is(equalTo(expected)));}
```

Key API: Assert

- `org.junit.Assert`
  - Contains a series of *static* "assertion methods."
    - » `assertThat( Object, org.hamcrest.Matcher )`
      - » Primitive-type value to be autoboxed.
      - » Just returns if two values (expected and actual values) match.
      - » Throws an AssertionError if two values do not match.
    - » `fail( String message )`
      - » Force to fail a test with a message.
      - » Throws an `AssertionError`.
    - » `assertTrue(boolean condition), assertFalse(boolean condition)`
      - » Asserts a condition is true/false.

- `assertEquals()` is deprecated in JUnit version 4.
  - It was a major assertion method until JUnit version 3.
  - Use `assertThat()` instead.
- Never use `assertEquals()` in your HW solutions.

Key API: CoreMatchers

- `org.hamcrest.CoreMatchers`
  - Contains static methods, each returning a matcher object that performs matching logic.
  - `is()`
    - » `assertThat(actual, is(expected))`
      - » Asserts "actual" and "expected" are equal.
    - » `assertThat(actual, is(nullValue()))`
    - » `assertThat(actual, is(notNullValue()))`
    - » `assertThat(actual, is(not(expected))`
      - » Asserts "actual" and "expected" are equal.
    - » `assertThat(actual, is(sameInstance(expected)))`
      - » Asserts "actual" and "expected" are identical instance with the same object ID.
    - » `assertThat(actual, is(instanceOf(Foo.class)))`
      - » Asserts "actual" is an instance of Foo.
        - » Foo may be a super class of "actual"'s class.

- » `assertThat(actual, containsString("foo"))`
- » `assertThat(actual, startsWith("foo"))`
- » `assertThat(actual, endsWith("foo"))`
- » `assertThat(actual, allOf(`
  `notNullValue(),instanceOf(Foo.class)))`
  - » Asserts all assertions hold.
- » `assertThat(actual, anyOf(`
  `containsString("HTTP/1.0"),`
  `containsString("HTTP/1.1")))`
  - » Asserts any of the assertions (at least one of the assertions) hold.

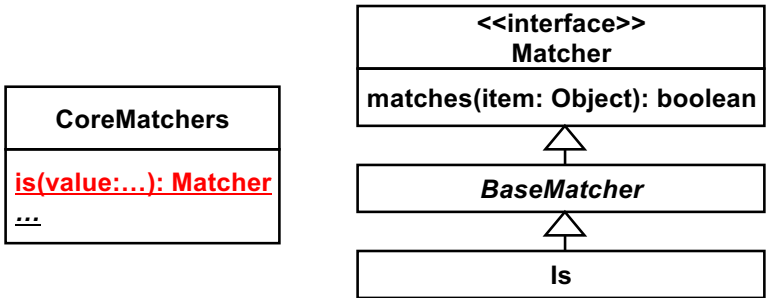
- `hasItem()`
  - » `ArrayList<String> actual = …;`
    `assertThat(actual, hasItem("Hello"));`
- `hasItems()`
  - » `assertThat(actual, hasItems("Hello", "World");`
- `everyItem()`
  - » `assertThat(actial, everyItem("Hello");`

```
String[] str = {"UMass Boston", "UMass Amherst"};
ArrayList<String> actual = Arrays.asList(str);
assertThat(actual, hasItem( containsString("UMass") ));
assertThat(actual, hasItem( endsWith("Boston") );
assertThat(actual, everyItem( containsString("UMass"));
```

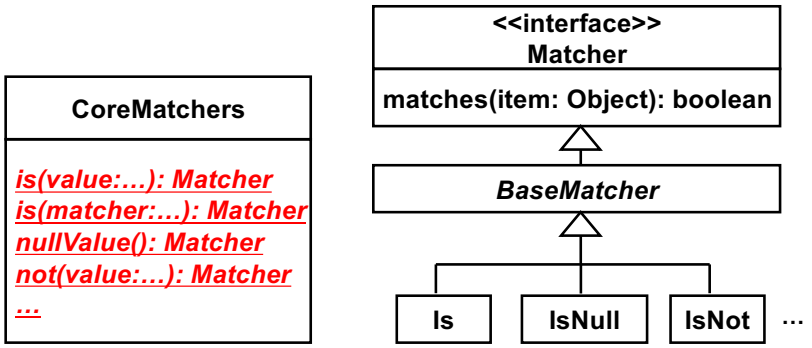
– It is important to learn what methods are available in `CoreMatchers` and what parameters the methods accept.
  » c.f. Javadoc API documentation.

Class Structure of JUnit APIs

- `org.junit.Assert`
  - `assertThat(Object, org.hamcrest.Matcher)`
- `org.hamcrest.CoreMatchers`
  - Contains static methods, each returning a matcher object that performs matching logic.
  - `is()`
    - » `assertThat(actual, is(expected))`



- `org.junit.Assert`
  - `assertThat( Object, org.hamcrest.Matcher )`
- `org.hamcrest.CoreMatchers`
  - `assertThat(actual, is(expected))`
  - `assertThat(actual, is(nullValue()))`
  - `assertThat(actual, is(notNullValue()))`
  - `assertThat(actual, is(not(expected))`

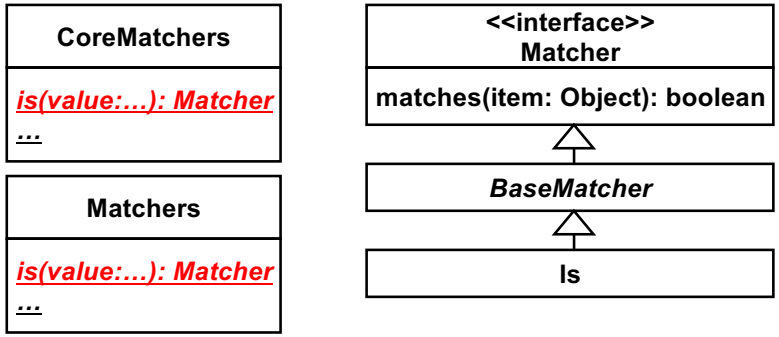


JUnit and Hamcrest

- Hamcrest provides many useful matchers for JUnit
  - junit.jar and hamcrest-core.jar from http://junit.org
  - hamcrest-all.jar from http://hamcrest.org
  - hamcrest-all.jar is a superset of hamcrest-core.jar.
    - If you use hamcrest-all.jar, you don't have to use hamcrest-core.jar.
      - No need to set both to CLASSPATH.

hamcrest-all.jar

- `org.hamcrest.CoreMatchers`
- `org.hamcrest.Matchers`
  - Contains static methods, each returning a matcher object that performs matching logic.
  - `Matchers` is a superset of `CoreMatchers`.



Matchers in hamcrest-all.jar

- Examine String data
  - `assertThat("UMass Boston", containsString("UMass"))`
  - `assertThat("UMass Boston", startsWith("UMass"))`
  - `assertThat("UMass Boston", endsWith("Boston"))`
  - `assertThat("UMass Boston", equalToIgnoringCase("UMASS BOSTON"))`
  - `assertThat("  UMass", equalToIgnoringWhiteSpace("UMass"))`
  - `assertThat(actual, isEmptyOrNullString())`
  - `assertThat("", isEmptyString())`
  - `assertThat("UMass Boston", stringContainsInOrder(`
    `Arrays.asList("UM", "Boston"));`
  -

- Examine numbers
  - `assertThat(10, is(10));`
  - `assertThat(10.3, closeTo(10, 0.3)); // PASS`
  - `assertThat(10, is(greaterThan(9));`
  - `assertThat(10, is(greaterThanOrEqualTo(10));`
  - `assertThat(10, is(lessThan(11));`
  - `assertThat(10, is(lessThanOrEqualTo(10));`

- Examine arrays
  - `String[] strArray = {"UMass","Boston"};`
    `assertThat(strArray, array( is("UMass"),`
    `startWith("B")));`
  - `assertThat(strArray, arrayContaining("UMass","Boston");`
  - `assertThat(strArray, arrayContainingInAnyOrder("Boston","UMass");`
  - `assertThat(strArray, arrayWithSize(2));`
  - `assertThat(strArray, is(not(emptyArray())));`
  - `assertThat(strArray, hasItemInArray("UMass"));`
  - `assertArrayEquals(strArray,strArray);`

- Examine collections
  - `String[] strArray = {"UMass","Boston"};`
    `ArrayList<String> actual = Array.asList(strArray);`
    `String[] expected = {"UMass","Boston"};`
  - `assertThat(actual, contains(expected));`
  - `assertThat(actual, contains("UMass", "Boston");`
  - `assertThat(actual, contains( is("Umass"),`
    `startWith("B")));`
  - `String[] expected2 = {"Boston","UMass"};`
    `assertThat(actual, containsInAnyOrder(expected2));`
  - `assertThat(actual, hasItem("UMass"));`
  - `assertThat(actual, hasItems("Boston")));`
  - `assertThat(actual, hasItems("UMass", "Boston")));`
  - `assertThat(actual, hasItem( containsString("Mass") ));`
    `assertThat(actual, hasItem( endsWith("Mass") );`
    `assertThat(actual, everyItem( containsString("s"));`

  - `String[] strArray = {"UMass","Boston"};`
    `ArrayList<String> actual = Array.asList(strArray);`
    `String[] expected = {"UMass","Boston"};`
  - `assertThat(actual, hasSize(2));`
  - `assertThat("UMass", isIn(strArray));`
  - `assertThat(actual, not(empty());`

- Examine maps
  - `HashMap<String, Integer> actual = new HashMap<String, Integer>();`
    `actual.put("foo", 0);`
    `actual.put("boo", 10);`
    `actual.put("bar", 100);`
  - `assertThat(actual, hasEntry("foo", 0));`
  - `assertThat(actual, hasEntry( endWith("oo"), greaterThan(5)) );`
  - `assertThat(actual, hasKey("bar"));`
  - `assertThat(actual, hasKey( startWith("b") ));`
  - `assertThat(actual, hasValue(0));`
  - `assertThat(actual, hasValue( lessThanOrEqualTo(100) ));`

Identity and Equality

- `assertThat(actual, is(sameInstance(expected)))`
  - Asserts "actual" and "expected" are ***identical*** instance with the same object ID.
    - » `assertThat(new Foo(), is(sameInstance(new Foo())));`
    - » `Foo f = new Foo();`
      `assertThat(f, is(sameInstance(f)));`
- `assertThat(actual, is(expected))`
  - A shortcut of `assertThat(actual, is(equalTo(expected)))`
- `assertThat(actual, is(not(expected)))`
  - A shortcut of `assertThat(actual, is(not(equalTo(expected))))`
  - Asserts "actual" is ***logically equal*** to "expected," as determined by calling `Object.equals(java.lang.Object)` on "actual."
    - » `actual.equals(expected);`

```
String str1 = "umb";
String str2 = "umb0".substring(0,2); // "umb0" -> "umb"
```

```
assertThat(actual, is(sameInstance(expected))); // FAIL
assertThat(actual, is(equalTo(expected)));      // PASS
```

- `equalTo()` calls `String.equals()` on "actual."
  - `String.equals()` overrides `Object.equals()` and returns true if two string values match.
  - c.f. Java API doc
- Note: `Object.equals(java.lang.Object)`
  - Implements the most discriminating possible equivalence relation on objects.
    - Returns true if two objects refer to the same instance (x == y has the value true): identity check.
    - c.f. Java API doc
  - Most pre-defined (API-defined) classes override equals() to perform appropriate equality check.


```
Date d1 = new Date(); //java.util.Date
Date d2 = new Date();
```

```
assertThat(actual, is(sameInstance(expected))); // FAIL
assertThat(actual, is(equalTo(expected)));      // PASS, most likely
```

- `equalTo()` calls `Date.equals()` on "actual."
  - `Date.equals()` overrides `Object.equals()` and returns true if two Date objects represent the same timestamp in millisecond.
  - c.f. Java API doc
- Most pre-defined (API-defined) classes override equals() to perform appropriate equality check.
- You need to override equals() in your own (i.e. user-defined) class, if you want to do equality check.

Equality Check for a User-defined Class

```
Person p1 = new Person("John","Doe");
Person p2 = new Person("John","Doe");
Person p3 = new Person("Jane", "Doe");
```

```
assertThat(p1, is(sameInstance(p1))); // PASS
assertThat(p1, is(sameInstance(p2))); // FAIL
assertThat(p1, is(equalTo(p2)));      // FAIL
```

- Person just inherits equals() from Object. The method just do identity check.
  - You need to override equals() in Person if you want equality check.

Person
- firstName: String - lastName: String
+ Person(first:String, last:String) + getFirstName(): String + getLastName(): String


```
Person p1 = new Person("John","Doe");
Person p2 = new Person("John","Doe");
Person p3 = new Person("Jane", "Doe");
```

```
assertThat(p1, is(sameInstance(p2))); // FAIL
assertThat(p1, is(equalTo(p2)));      // PASS
assertThat(p1, is(sameInstance(p3))); // FAIL
assertThat(p1, is(equalTo(p3)));      // FAIL
```

Person
- firstName: String - lastName: String
+ Person(first:String, last:String) + getFirstName(): String + getLastName(): String + equals(anotherPerson:Object): boolean

```
if( this.firstName.equals(((Person)anotherPerson).getFirstName())
    && this.lastName.equals(((Person)anotherPerson).getLastName())){
  return true;
}
else{
  return false;
}
```

Alternatively…

```
Person p1 = new Person("John","Doe");
Person p2 = new Person("John","Doe");
Person p3 = new Person("Jane", "Doe");
```

```
assertThat(p1, is(sameInstance(p2)));                              // FAIL
assertThat(p1.getFirstName(), is(equalTo(p2.getFirstName())));     // PASS
assertThat(p1.getLastName(), is(equalTo(p2.getLastName())));       // PASS

assertThat(p1, is(sameInstance(p3)));                              // FAIL
assertThat(p1.getFirstName(), is(equalTo(p3.getFirstName())));     // FAIL
assertThat(p1.getLastName(), is(equalTo(p3.getLastName())));       // FAIL
```


One More Method in Assert

- `org.junit.Assert`
  - `assertArrayEquals(expecteds, actuals)`
    - » Assert two arrays are equal (i.e. all element values are equal in the two arrays)
      - » Can accept an primitive-type arrays and Object arrays
        - » Primitive types: boolean, byte, char, double, float, int, long, short
        - » You can pass an array of arrays (multi-dimensional array) to `assertArrayEquals(Object[], Object[])`.
  - Assert has some extra methods, but you don't really have to learn/use them.


```
int[] i1 = {2,0,0,0};
int[] i2 = {2,0,0,0};
assertArrayEquals(i1, is(i2)); // PASS
```

```
String[] str1 = {"UMass","Boston"};
String[] str1 = {"UMass","Amherst"};
assertArrayEquals(str1, is(str2)); // FAIL
```

```
Person[] persons1 = {new Person("Mickey", "Mouse"),
                     new Person("Minnie", "Mouse")};
Person[] persons2 = {new Person("Mickey", "Mouse"),
                     new Person("Hanna", "Suzuki")};
```

```
assertArrayEquals(persons1, persons2); // PASS if...
```

```
assertThat(new Person("Mickey","Mouse"),
           is(new Person("Mickey","Mouse"))); // PASS if ...
```

Person
- firstName: String - lastName: String
+ Person(first:String, last:String) + getFirstName(): String + getLastName(): String

Tips for Unit Testing

- In principle, you should write a unit test(s) for each public method of your class.
- However, methods with very obvious functionalities/behaviors do not need unit tests.
  - e.g. simple getter and setter methods
  - unless they behave in some unique/interesting/complex way.
    - e.g. getting some data from an external entity (e.g. DB)
- Write a unit test whenever you feel you need to comment the behavior of a method.

Benefits of Unit Tests

- Can test classes and their methods thoroughly.
  - Provide you a great confidence and in turn satisfaction.
- Can trigger/motivate design changes
  - You as a programmer are the first "user" of your own code.
  - If you feel your class/method is not easy to use, that encourages you to revise the current design.

- Can be useful as sample code to use your class/method (the best sample code, in fact)
  - No need to write sample use cases and sample code in API documentations and other docs.
  - When you forgot how to use a class/method you implemented.
  - When you use a class/method that someone else implemented.

Continuous Unit Testing

- You as a programmer do it *continuously*
  - as you write code and whenever you revise existing code
  - Code-test-code-test, rather than code-code-code-test
  - Test-code-test-code
    - "Test first": Test-driven development (TDD)
- Goal: Continuously make sure that your code works as expected and gain strong confidence and a peace of mind about your code.

- Test your code *early*, *automatically* and *repeatedly*.
  - To maximize the benefits of unit testing.
- Early testing
  - Do coding and unit testing at the same time.
- Automated testing
  - Run ALL test cases in an automated way.
    - Never think of selecting and running test cases by hand.
- Repeated testing
  - Run ALL test cases whenever changes are made in the code base.

Benefits of Continuous Testing

- Can perform *regression testing* through continuous unit testing
  - Regression
    - A bug that emerges as a by-product in making changes in the code base
      - e.g., adding new code to the code base or revising existing code in the code base.
  - Regression testing
    - Uncovering regressions after changes are made in the code base
  - Seamlessly integrate unit testing and regression testing
- *Immediate*ly giving feedback on regressions to development project members and fix them.
  - DO: Code → test → small regression fixes → test
  - DON'T: Code → code → code → test → big regression fixes
    - The amount of regressions (and the cost to fix them) can exponentially increase as time goes without continuous testing.

HW 5

- Write test cases for the code you wrote in HW2 (polygon example)
  - Test two subclasses of Polygon
    - Triangle and rectangle
  - Write at least one test case for every single method.
- Turn in a build script, src and test/src for each.
  - The script should build all source code, run all text cases automatically.

- Use <batchtest> to have Ant search test classes in your project directory and run all of them (RectangleTest and TriangleTest).
  - <junit …>

```
…
<batchtest …>
...
</batchtest>
...
</junit>
```

  - c.f. JUnit documentation