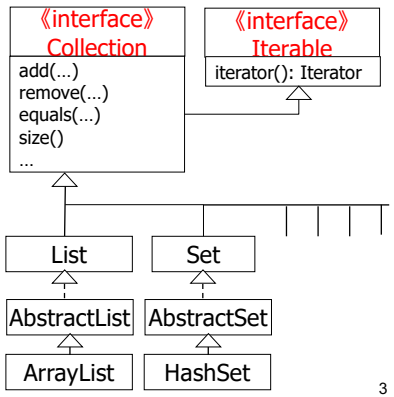
JUnit API (cont'd)

Assertions.assertIterableEquals()

- `org.junit.jupiter.api.Assertions`
  - `assertIterableEquals( Iterable expected, Iterable actual )`
    - Returns if expected and actual iterables are deeply equal (i.e., if they have equal elements in the same order).
      - »
```
List<String> l1 = List.of("UMass", "Boston");
List<String> l2 = List.of("UMass", "Amherst");
assertIterableEquals(l1, l2);  // NOT PASS
```
      - » c.f. assertArrayEquals()
    - » Throws an `org.opentest4j.AssertionFailedError` if two iterables are not equal.
      - JUnit catches it; your test cases don't have to.
    - JUnit judges that a test method (test case) passes if it normally returns without `AssertionFailedError`

Iterable

- `Collection`
  - The root interface for all collection classes
- `Iterable`
  - The super interface of `Collection`.
  - All collection classes implements it.
- `assertIterableEquals()` takes any collections.



Assertions.assertIterableEquals()

- `assertIterableEquals( Iterable expected, Iterable actual )`
  - Returns if expected and actual iterables are deeply equal.
    - They don't have to be of the same type.

```
List<String> l1 = Arrays.asList("UMass", "Boston");
List<String> l2 = Arrays.asList("UMass", "Boston");
assertIterableEquals(l1, l2);  // PASS
```

```
ArrayList<String> al = new ArrayList<>(
                                   Arrays.asList("UMass", "Boston"));
LinkedList<String> ll = new LinkedList<>(
                                   Arrays.asList("UMass", "Boston"));
assertIterableEquals(al, ll);  //PASS
```

Recap: `PrimeGenerator`

Generates prime numbers in b/w `from` and `to`.

```
Class PrimeGenerator {
  protected long from, to;
  protected LinkedList<Long> primes;

  public void generatePrimes(){ ... }
  public LinkedList<Long> getPrimes(){ return primes };
  ...
```

Tests

```
PrimeGenerator gen = new PrimeGenerator(1, 10);
gen.generatePrimes();
Long[] expectedPrimes = {2L, 3L, 5L, 7L};
assertArrayEquals( expectedPrimes,
                   gen.getPrimes().toArray() );
```

```
assertIterableEquals( Arrays.asList(expectedPrimes),
                      gen.getPrimes() );
```


Equality and Identity

- `assertEquals( Object expected, Object actual )`
  - Asserts that `actual` is *logically equal* to `expected`
    - » By calling `expected.equals(actual)`.
      - » c.f. `Object.equals()`

- `assertSame( Object expected, Object actual )`
  - Asserts that `expected` and `actual` refer to the *identical object*
    - » by checking if `expected.hashCode()==actual.hashCode()`

```
» Foo f = new Foo();
  assertSame(f, f);                    // PASS
```

```
» Singleton instance1 = Singleton.getInstance();
  Singleton instance2 = Singleton.getInstance();
  assertSame(instance1, instance2);  // PASS
```



```
String str = "umb";          // Syntactic sugar for
                             //     String str = new String("umb");
                             // str contains a pointer (or reference) to
                             // the String instance.
String expected = str;       // expected and actual refer to the
String actual = str;         // identical String instance.
```

```
assertSame(expected, actual);      // PASS
assertEquals(expected, actual);    // PASS
```

- `assertSame()` checks whether
  - `expected.hashCode()==actual.hashCode()` is true.

- `assertEquals()` checks whether
  - `expected.equals(actual)` returns true.
    - `String.equals()` overrides `Object.equals()` and returns true if two String instances contain the same String value.


```
String expected = "umb";    // Syntax sugar for:
                            //    String expected = new String("umb");
String actual = "umb0".substring(0,2);
                            // Syntax sugar for:
                            //    String temp = new String("umb0");
                            //    actual = temp.substring(0, 2);
                            // "umb0" -> "umb"
                            // expected and actual refer to
                            // different String instances.
```

```
assertSame(expected, actual);    // FAIL
assertEquals(expected, actual);  // PASS
```

- `assertSame()` checks whether
  - `expected.hashCode()==actual.hashCode()` is true.

- `assertEquals()` checks whether
  - `expected.equals(actual)` returns true.
    - `String.equals()` overrides `Object.equals()` and returns true if two String instances contain the same String values.

HW 2

- Revise `PrimeGenerator` to be a *singleton* class.
  - Name the revised class as `SingletonPrimeGenerator`.
  - Add a static factory method: `getInstance()`.
  - Remove the public constructor.
  - Have your class receive `from` and `to` values, as you like.

- Write a test class (`PrimeGeneratorTest`) with JUnit
  - Verify `getInstance()` returns a non-null value.
    - Use `Assertions.assertNonNull()`
  - Verify `getInstance()` returns the identical instance when it is called multiple times.
    - Use `assertSame()`
  - Verify `getPrimes()` returns the expected result.
    - Use `assertIterableEquals()`
  - Verify your class throws an expected exception when wrong ranges (`from-to` pairs) are given
    - e.g., [-10, 10], [1, 10], [100, 1]

Recap: Equality

- `assertEquals( Object expected, Object actual )`
  - Asserts that `actual` is *logically equal* to `expected`
    - » By calling `expected.equals(actual)`.
      - » c.f. `Object.equals()`
  - Checks whether
    - `expected.equals(actual)` returns true.
      - `String.equals()` overrides `Object.equals()` and returns true if two String instances contain the same String values.

Object.equals()

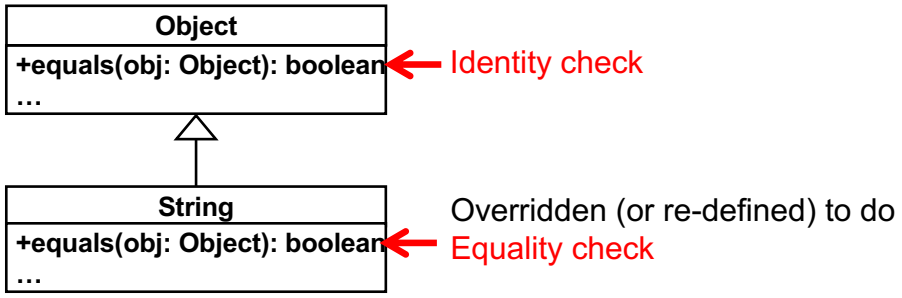
- `Object.equals(Object obj)` compares two objects with:
  - `if( this.toString()==obj.toString() ){ return true; } else if{ return false; }`
  - `Object.toString()` returns the identity of an object.
    - String data that consists of an object ID, a class name and a package name.
      - e.g., edu.umb.cs680.junit5intro.Calculator@2b2948e2
  - Performs identity check (not equality check).
    - Even though the method name says “equals.”

`Object.equals()`

- `Object.equals(Object obj)` compares two objects with:
  - `if( this.toString()==obj.toString() ){ return true; }`
    `else if{ return false; }`
  - `Object.toString()` returns the identity of an object.
    - String data that consists of an object ID, a class name and a package name.
      - e.g., edu.umb.cs680.junit5intro.Calculator@2b2948e2
    - Performs identity check (not equality check).
      - Even though the method name says "equals."
- Most Java API classes (e.g. String) override `Object.equals()` to perform appropriate equality check.
  - However, user-defined classes DO NOT… to be discussed.

`equals()` in Java API

- Most Java API classes override `Object.equals()` to perform appropriate equality check.
  - e.g., `String` overrides `Object.equals()` and returns true if two String instances contain the same String values.



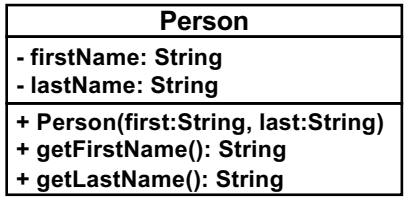
Read the source code of `String.equals()` if you are interested.

Equality Check for User-defined Classes

- When you define your own class, it is implicitly treated as a subclass of `Object`
  - Your class inherits `Object.equals()`.
- Your class's `equals()` does identity check by default
  - Unless you override `equals()`.


```
Person p1 = new Person("John","Doe");
Person p2 = new Person("John","Doe");
Person p3 = new Person("Jane", "Doe");
assertSame(p1, p1);          // PASS
assertSame(p1, p2);          // FAIL
assertEquals(p1, p2);        // FAIL
assertEquals(p1, p3);        // FAIL
assertEquals(p2, p3);        // FAIL
```

- `Person` inherits `Object.equals()`. The inherited method performs *identity check* by default for `Person` instances.
  - You need to override `equals()` in `Person` if you want equality check.



```
Person p1 = new Person("John","Doe");
Person p2 = new Person("John","Doe");
Person p3 = new Person("Jane", "Doe");
assertSame(p1, p2);                 // FAIL
assertEquals(p1, p2);               // PASS
assertSame(p1, p3);                 // FAIL
assertEquals(p1, 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;
}
```


- Define `equals()` in `Person`, if your team has a consensus about the equality of `Person`s.
- If the consensus may often change, or if there is no reasonable consensus…
  - you should craft equality-check logic in your test class, not in `Person`.

```
Person p1 = new Person("John","Doe");
Person p2 = new Person("John","Doe");
Person p3 = new Person("Jane", "Doe");
assertEquals(p1.getFirstName(), p2.getFirstName());     // PASS
assertEquals(p1.getLastName(), p2.getLastName());       // PASS

assertNotEquals(p1.getFirstName(), p3.getFirstName()); // PASS
assertNotEquals(p1.getLastName(), p3.getLastName());   // PASS
```

- JUnit judges that a test method (test case) passes if it normally returns (i.e., if all four assertion methods return) without `AssertionFailedError`

How to Write Equality-check Logic

- As you use more information for an equality check, you need to call assertion methods more often in a single test method.
  - e.g., first and last names, DOB, zip code for home address.
    - Need to call `assertEquals()` 4 times.
  - e.g., car name, manufacturer name, production year
    - Need to call `assertEquals()` 3 times.
- Equality-check logic gets less clear.
- In general, it makes more sense to perform equality-check by calling assertion methods less often.
  - Consider a String-to-String or array-to-array comparison.

String-to-String Comparison

```
@Test
... checkPersonEqualityWithJohnJane(){
  Person p1 = new Person("John","Doe",
                         LocalDate...,
                         02125);
  Person p2 = new Person("Jane", "Doe",
                         LocalDate...,
                         02125);

  assertEquals(p1.getFirstName(),
               p2.getFirstName());
  assertEquals(p1.getLastName(),
               p2.getLastName());
  assertEquals(p1.getDOB(),
               p2.getDOB());
  assertEquals(p1.getZipCode(),
               p2.getZipCode());
}
```

```
private String eol =
    System.getProperty("line.separator");

private String personToString(Person p){
  return p.getFirstName() + eol +
         p.getLastName() + eol +
         p.getDOB().toString() + eol +
         p.getZipCode() + eol; }

private String concatenamePersonInfo(
                                String[] p){
  String personInfo;
  for(String info: p){
      personInfo += info + eol;} }

@Test
... checkPersonEqualityWithJohnJane(){
  String[] expectedArray =
      {"John", "Doe", ..., "02125"};
  String expected =
      concatenatePersonInfo(expectedArray);

  Person actual = new Person(
                "John","Doe", ..., 02125);

  assertEquals(expected,
               personToString(actual) ); }
```

Array-to-Array Comparison

```
private String[] personToStringArray( Person p ){
  String[] personInfo = {p.getFirstName(),
                         p.getLastName(),
                         p.getDOB().toString(),
                         p.getZipCode() };
  return personInfo;
}

@Test
public void checkPersonEqualityWithJohnJane(){
  String[] expected = {"John", "Doe", ..., "0215"};

  Person actual = new Person("John","Doe", ..., 02125);

  assertArrayEquals(expected,
                    personToStringArray(actual) );
}
```


HW 3

- Define the `Car` class and implement its getter methods.
  - ```
    public class Car {
      private String make, model;
      private int mileage, year;
      private float price;  }
    ```

- Write a test class (`carTest`) with JUnit
  - Include a private method `carToStringArray()`
  - Define a test method `verifyCarEqualityWithMakeModelYear()`
    - Create two `Car` instances and check their equality with array-to-array comparison
      - Use make, model and year in equality-check logic

      - ```
        String[] expected = {"Toyota", "RAV4", "2018"};
        Car actual = new Car(...);
        assertArrayEquals(expected,
                          carToStringArray(actual) );
        ```