CS 5004 Module 2 Worksheet
Methods & Exceptions

Each worksheet is to help you remember and understand the materials contained in each module so that you can get the most out of your experience. We start by asking questions that are covered in the module and then, in the second part, ask you to apply this knowledge. By taking the time to answer each question carefully, these worksheets can serve as the basis for review for the more formal assessments in this class.

1. Is a variable of type `` `float` `` the best type to represent the price of a Book? Why or why not?

The float and double types are particularly ill-suited for monetary calculations because it is impossible to represent 0.1 (or any other negative power of ten) as a float or double exactly. It is because floats and doubles cannot accurately represent the base 10 multiples that we use for money.

2. What is the recipe that should be used to define a method in a class?

We used all the instance variables and field data to define a method in class. We also need to add some signature and comments before the method so that we know what this method returns and what it is used for.

3. Java is a statically-typed language that requires that you explicitly declare types. Which parts of a method signature need an explicit type declaration?

In the head line of the method signature, we need to explicitly indicate what type this method will return, for example void means it returns nothing, String means it returns a String.

4. `` `toString` `` is one of the methods that every class implements. What is its signature?

```
public String toString() {
    return String data;
}
```

5. `` `String.format` `` method uses specific format specifiers to create formatted strings. For each of the following, indicate the type that is being used and what the format specifier means.

%d	Formats the argument as a decimal integer.

`%10d`	If no. of digits is less than 10, prints output padded to the left
`%f`	Formats the argument as a float.
`%.3f`	Float with precision to three decimal places
`%b`	If the argument arg is null, then the result is "false".
`%c`	The result is a Unicode character
`%s`	If arg implements Formattable, then arg.formatTo is invoked. Otherwise, the result is obtained by invoking arg.toString().

6. Consider the comment that was used video for the return of the `` `toString` `` method:

```
/**
 * Returns a string representation of this person with first and last name.
 *
 * @return a formatted string
 */
public String toString() {
  // implementation omitted
}
```

Why is this a poorly written comment?

It is because the formatted string is too vague to describe what the method will return. It should specify what is the new formatted string.

7. When using `` `assertEquals` `` with floating point numbers, you are required to pass three parameters. What is the third parameter and why do you need it?

The third parameter we put in is the precision indicator. Since it is hard to compare two float numbers directly due to precision errors (15.9999999 is not equal to 16). So, this version checks if the expected value (first parameter) is equal to the actual value (second parameter) within the error threshold passed as the third parameter.

8. When comparing String variables, why do we need to use the `` `equals` `` method instead of the == operator?

We can use == operators for reference comparison (address comparison) and .equals() method for content comparison. In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects. In Java, the String equals() method compares the two given strings based on the data/content of the string. If all the contents of both the strings are the same, it returns true. If all characters are not matched, then it returns false.

9. When implementing `sameAuthor` in the Book class, we *delegated* the responsibility of comparing the author object to the Person class. What does that mean and why is it a *good thing*?

Since it is hard for Book class to access private variables in the Person class. We can define a public method in the Person class to compare two person and their variables and use the public method of Person in the Book class.

10. Suppose you have a class called `University`, how would you allocate an instance of this class if you were told that the constructor of the class took a single string containing the name of the university. Create an object for Northeastern University as an example.

    University NEU = new University("Northeastern University");

11. Consider the following constructor that was part of the `Person` class. Add the code that you would need to validate that the `year` parameter represents someone who was born after 1920.

```
/**
 * Constructs a Person object.
 *
 * @param firstName   the first name of this person
 * @param lastName    the last name of this person
 * @param yearOfBirth the year of birth of this person
 */
public Person(String firstName, String lastName, int yearOfBirth) throws
IllegalArgumentException {

  if (yearOfBirth < 1920) {
      throw new IllegalArgumentException("Person must be born after 1920.");
  }

  this.firstName = firstName;
  this.lastName = lastName;
  this.yearOfBirth = yearOfBirth;
}
```

12. Add a test to the following method that would be able to test that the method throws an exception when the parameter is invalid:

```
public class PersonTest {

  private Person john;
  @Before
  public void setUp() {
    john = new Person("John", "Doe", 1945);
  }


  @Test (expected = IllegalArgumentException.class)
 Public void testIllegalYearOfBirth() {
   Person joe = new Person(“Joe”, “doe”, 1903);
  }

}
```

13. Add a test to the following method that would be able to test that the method does not throw an exception when the parameter is valid:

```
public class PersonTest {

  private Person john;

  @Before
  public void setUp() {
    john = new Person("John", "Doe", 1945);
  }


 @Test
 Public void testYearOfBirth() {
   Int birthyear = john.getyearOfBirth();
   assertEquals(1945, birthyear);
 }

}
```

14. How do we document that the method throws an exception in the Javadoc for the method?

```
/**
 * Constructs a Person object.
 *
 * @param firstName   the first name of this person
 * @param lastName    the last name of this person
 * @param yearOfBirth the year of birth of this person
 * @throws IllegalArgumentException if a less than 1920 year of birth is passed as
* an * argument
 */
public Person(String firstName, String lastName, int yearOfBirth) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.yearOfBirth = yearOfBirth;
}
```

15. What are guidelines for using exceptions?

- Write the purpose statement
- Write a method signature
- Using the template write the method body so that it works correctly under ideal circumstances.
- Carefully think about all situations that can occur, other than ideal circumstances. This includes possibly invalid inputs, invalid computations, or results. These are possible errors.
- If there is a way to prevent the error from happening, do it. This is the best kind of error handling. Else go to the next step.
- If there is a way to recover from the error in this method itself, do it. There is no need to use exceptions as the recovery happens in the same method as the error. Else go to the next step.
- Choose an appropriate exception type for the error. Declare that this method throws this exception in its signature, and throw this exception appropriately.

16. How does Java decide what method to call when two methods have the same name in different classes?

If the two methods are in different classes, then the class of the object decides which method to call. This is also called dynamic dispatch.

17. How does Java decide what method to call when two methods have the same name in the same class?

Java allows multiple methods in the same class to have the same names as long as they have different signatures. So, if two methods have the same name in the same class, java need to see what is the signature and what is the method takes in to decide which method will be used.

18. What is the difference between a static variable and an instance variable?

A static variable is a single copy of variable is created and shared among all objects at the class level. An instance variable can be changed by other methods, but static variable cannot be changed.

19. What is the preferred way to call a static method?

A static method can be called directly from the class, without having to create an instance of the class. A static method can only access static variables; it cannot access instance variables. Since the static method refers to the class, the syntax to call or refer to a static method is: class name. method name.

20. Suppose there is class that has the following definition, write an appropriate constructor that has two parameters one of type `int` and one of type `char`

```
public class YourClass {
  private int information;
  private char moreInformation;


/**
 * Constructs a yourclass object.
 *
 * @param information   the information of this class
 * @param moreInformation    the more information of this class
 */
 public YourClass(int information, char moreInformation) {
      this.information = information;
      this.moreInformation = moreInformation;
 }



}
```

21. What is a *no-argument constructor?* Does every class have a no-argument constructor? What is a default constructor?

No-argement constructor means the constructor takes no argument into the class.
Yes, every class have a no-argument constructor.
A constructor is called "Default Constructor" when it doesn't have any parameter.

22. Articulate what each of the following calls used in a test method does:

`assertEquals(...)`	checks that the two objects are equals or not.
`assertTrue(...)`	checks whether the expected value is true or not.
`assertFalse(...)`	checks whether the expected value is false or not.
`fail(...)`	The fail assertion fails a test throwing an AssertionError unconditionally.