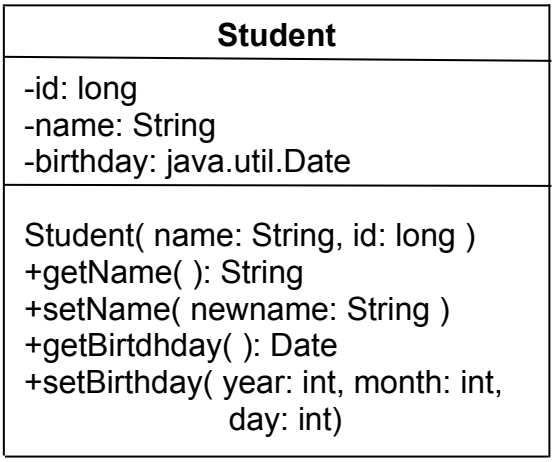
A Student has a name, id, and birthday as shown in this *UML Class Diagram*.



1. Complete the class declaration, and declare the attributes of a Student

```
import _____________________;
_____________ _____________ Student {
    _____________ _____________________________ ;
    _____________ _____________________________ ;
    _____________ _____________________________ ;
```

2. Write an *accessor* method for the Student's name attribute.

```
public _____________ _____________( ) {


}
```

3. Write an *mutator* method for the Student's name attribute. Do not accept a new name (the parameter) if it is null or an empty string. In that case, do nothing (better solution is to throw an Exception, which we haven't studied yet.)

```
public _____________ _____________ ( String _____________ ) {



}
```

4. Write a *constructor* so that we can initialize students like this:

```
Long id = 5810123456L;
Student pee = new Student("Pirawat", id);
```

5. Write an `equals` method that returns true of two students have the same **`id`**. Complete the Javadoc.

Use the **4-step pattern for equals**: 1) test if parameter is null, 2) test parameter reference is same type as this object (`x.getClass()`, 3) *cast* the parameter to a reference of type Student, 4) compare attributes as required. Also complete the Javadoc.

```
/**
 * Test if two students are equal.
 * _____________ obj is another object to compare to this one
 * _____________ true iff obj is a Student with same id as this Student
 */
public boolean equals( Object  obj ) {

     ____________________________________________________________;

     ____________________________________________________________;

     ____________________________________________________________;

     ____________________________________________________________;

     ____________________________________________________________;
}
```

6. Correct this **`toString`** method. Write your changes in the code.

```
/** Return a string representation, such as "Cat [5610541234]" */
public void toString( ) {
     System.out.println( this.name + " [" + this.id + "]");


}
```

7. Our Student class has a birthday attribute with "get" and "set" methods:

```
public class Student {
    private Date birthday;
    /** Get the student's birthday. */
    public Date getBirthday( ) {
        return this.birthday;
    }
    /** Set the student's birthday. */
    public void setBirthday(int year, int month, int day) {
        // Date constructor is weird.
        // year value is year-1900, e.g. Year 2000 is 100
        // month value is 0=January, 1=February, ..., 12=Decemeber
        this.birthday = new Date(year-1900, month-1, day);
    }
```

However, Java **discourages** the use of the Date class and recommends using LocalDate instead.

We cannot change the ***method signatures*** for getBirthday() and setBirthday() because other classes are using those methods! *"Method signature"* means how the method appears: its name, parameters, visibility, and return type.

Fortunately, our code ***encapsulates*** *and* ***hides*** the birthday attribute.

Therefore, we can *change the internal implementation* as long as we don't change the method signatures.

Modify the code so that the Student's birthday is a java.time.LocalDate instead of Date.

a) modify **`getBirtthday()`** to create a new Date object and return it. **`LocalDate`** has these methods:

getYear() returns the year

getMonthValue() returns the month as an integer. 1 = January, 12 = December

getDayOfMonth( ) returns the day of the month

Use these methods to create a new Date object and return it.

b) modify **`setBirthday`**() so that it updates birthday as a LocalDate object instead of Date.

LocalDate has a *factory method* to create a new object: LocalDate.of(year, month, day)

Write your solution in BlueJ first so you get the syntax correct; then write it here:

```
import java.util.Date;
import java.time.LocalDate;
public class Student {
    private LocalDate birthday;
    /** Get the student's birthday. */
    public Date getBirthday( ) {
        _____________________________________________________________
        _____________________________________________________________
    }
    /** Set the student's birthday.
     * @param year is the year of birth, e.g. 2001.
     * @param month is the birth month, 1=January, .., 12=December.
     * @param day is the day of the month.
     */
    public void setBirthday(int year, int month, int day) {
        _____________________________________________________________
        _____________________________________________________________
    }
```

8. For each item in the left column, identify the kind of thing is represents using items in the right column.

`char`	attribute of an *object*
`Character`	class
`List`	instance method
`System`	interface
`java.lang`	package
`java.lang.System`	primitive type
`length() in "Harry".length()`	static attribute (attribute of a *class*)
`org.junit`	static constant
`java.lang.Math.PI`	static (class) method
`java.lang.Math.sqrt( )`	variable name
`System.in`	
`System.in.read()`	
`LocalDate.now( )`	
`next() in Scanner class`	

9. Which *package* contains each of these classes?

The choices are `java.io`, `java.lang`, `java.time`, and `java.util`.

_____ `InputStream`

_____ `File`

_____ `PrintStream`

_____ `Math`

_____ `Double`

_____ `String`

_____ `Collections`

_____ `ArrayList`

_____ `Date`

_____ `LocalDate`

_____ *`Runnable`*

_____ `System`

10. Name the *interface* that defines these methods. If more than one possible interface, choose the interface at the base of an interface hierarchy.

	`run()`

	`hasNext()`
	`next()`
	`remove()`

	`compareTo(T other)`

	`add(T obj)`
	`contains(T obj)`
	`remove(T obj)`
	`size()`

	`add(T obj)`
	`contains(T obj)`
	`get(int index)`
	`remove(int index)`
	`size()`

11. Name the exception that is thrown by each of these:

	`String[] animals = {"cat", "dog"};` `System.out.println(animals[2]);`

	`List<String> list =`
	`  Arrays.asList(animals);`
	`// set replaces an existing element`
	`list.set(2, "bat");`

	`InputStream in = new`
	`  FileInputStream("does not exist");`
	`int c = in.read();`

	`// use Scanner to parse a string`
	`Scanner s = new Scanner("hi there");`
	`s.next();  // "hi"`
	`s.next();  // "there"`
	`s.next();  // ?`

	`int n = 7/8;` `int m = 1/n;`

	`// Integer & Double extend Number`
	`Number n = new Integer(10);`
	`Double d = (Double) n;`

12. Write example code that will throw a **`NullPointerException,`** using at most 3 statements. Use only classes in the Java API.

13. Write example code that clearly shows polymorphism being used. Use only classes in the Java API, and indicate where in your code shows polymorphism occurring. It must be something that we can type into jshell and run (no credit if it doesn't work). Please make the example short -- no longer than 5 statements.

14. Its important to know some commonly used classes for any programming language. Otherwise, you waste a lot of time trying to find something you need or (worse) writing code for something the language API already provides.

Name each of these Java classes.

	Provides access to operating system resources, such as environment variables, console input, and output to the console.
	Utility methods for working with collections, including sorting and searching. Can also reverse the order of a list or create an immutable *view* of an existing List.
	Utilities methods for working with arrays, including sorting, searching, copying arrays, and filling an array with a constant value.
	Common mathematical functions like square root.
	A wrapper to encapsulate a double as an object. Also has constants for the smallest and largest possible values, Infinity, and a method to get a double value from a String.
	A mutable string type that you can append to, modify its contents, etc. Useful for building or editing Strings.
	A class that lets an application break a String into tokens (words), but not Scanner. This class is much faster than scanner for splitting a String into tokens.