DAILY ASSESSMENT FORMAT

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **11/06/2020** | **Name:** | **DIVYASHREE LV** |
| **Course:** | **KiCad like a pro 2nd edition** | **USN:** | **4AL17EC030** |
| **Topic:** | **Start a new project, netlist and footprint association and placing** | **Semester & Section:** | **6TH Sem ‘A’ sec** |
| **FORENOON SESSION DETAILS** | | | |
| **Image of session** | | | |

# Report :

1. **Creating Your Library**

Open KICAD, and we’ll be greeted with the **Control Panel**.

* 1. Next, select **File » New Library** to open **Library Window**.
  2. Before creating any new parts, you need to save library by selecting **File » Save** (or

**Cmd + S** on Mac and **Ctrl + S** on Windows)

* 1. Now that your library is saved, you just need to activate it. Go back to **Control Panel**, right-click your new library, and select **Use**.

# Create PCB footprint component

The blank canvas of the new footprint editor.

You can also launch the footprint editor from the main Kicad window. To do this, go to the main Kicad window and click on the fourth button from the right. Notice that it contains the same icon as the one in Eeschema, with the IC and a pencil over it.

Creating the new footprint. We need to add boundaries and pins.

Move the two text labels so that there is enough space between them for the boundary. Use the M key for this, just like you moved components in Eeschema.

Let’s continue with the holes for the pins. We need eight holes. We need to select holes that are wide enough so that the pins of the actual component will fit through them and that the exact distance from the adjoining pins as we measured earlier. So, let’s start with a pitch. We know that the pitch (distance) between the pins is 2.54 millimeters, so to make it easy to space the pins at this exact distance, we’ll set the grid to be this size**.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **11/06/2020** | **Name:** | **DIVYASHREE LV** |
| **Course:** | **Java** | **USN:** | **4AL17EC030** |
| **Topic:** | **Programming core java** | **Semester& Section:** | **6th sem & ‘A’ sec** |

|  |
| --- |
| **AFTERNOON SESSION DETAILS** |
| **Image of session:** |
|  |
|  |

# Report:

**toString() method:**

If you want to represent any object as a string, **toString() method** comes into existence. The toString() method returns the string representation of the object.

If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.

# Inheritance:

Inheritance is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs

# Encapsulation:

* Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.
* In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding**.**

# Generic Methods

* + All generic method declarations have a type parameter section delimited by angle brackets (< and >) that precedes the method's return type ( < E > in the next example).
  + Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.
  + The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.
  + A generic method's body is declared like that of any other method. Note that type parameters can represent only reference types, not primitive types (like int, double and char).

# Java Type Casting

Type casting is when you assign a value of one primitive data type to another type. In Java, there are two types of casting:

* Widening Casting (automatically) - converting a smaller type to a larger type size byte -> short -

> char -> int -> long -> float -> double

* Narrowing Casting (manually) - converting a larger type to a smaller size type double -> float -> long -> int -> char -> short -> byte