**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **12/06/2020** | **Name:** | **Lavanya B** |
| **Course:** | **Kicad** | **USN:** | **4al17ec043** |
| **Topic:** | **PCB designing** | **Semester & Section:** | **6th A** |
| **Github Repository:** | **Lavanya-B** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report**  **Creating PCB footprint component**  **Create a PCB footprint**  **A footprint describes the interface between the circuit board and the component it self. This is often called the land-pattern. At the least it needs to contain so called pads. It is also suggested to at least include the part outline and part identifiers on the silk and fab layers. If a footprint is specialized for a single component then include the part number and the manufacturer name. For generic footprints include all identifying parameters.**  **Add footprint search path**    **In KiCad, one can define some paths using an environment variable. A few environment variables are internally defined by KiCad, and can be used to define paths.**  **This is useful when absolute paths are not known or are subject to change. This is the case for "official" libraries built for KiCad:**  **-for the path of these libraries, when installed on your disk**  **-for the path of 3D shapes files used in footprint definitions.**  **For instance, the full path of connect.pretty footprint library is defined like this, when using the KISYSMOD environment variable to define the full path: ${KISYSMOD}/connect.pretty**  **Obviously, one can use a usual full path definition, if this full path is well known, and never changes.**  **Production file**  **The Gerber format is an open 2D binary vector image file format. It is the standard file used by printed circuit board (PCB) industry software to describe the printed circuit board images: copper layers, solder mask, legend, etc.**  **Change to the "PCB" edit mode on your Fritzing. And click the "Export for PCB" ▼ button on the lower toolbar. Make sure you click the arrow for other format options, not the main button, otherwise, Fritzing will export to PDF by default.**  **The data type selection menu will open. Click "Extended Gerber (RS-274x)**  **The "Choose a folder for exporting" dialog will open. Create a new folder and select it (I made the "gerber-files" folder.) The Gerber files and drill file will then be exported into the chosen folder.**  **Open the folder and delete the "Countdown\_pnp.txt". It is not needed for PCB manufacture.** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **12/06/2020** | **Name:** | **Lavanya B** | |
| **Course:** | **JAVA** | **USN:** | **4al17ec043** | |
| **Topic:** | **Programming core JAVA** | **Semester & Section:** | **6th A** | |
| **AFTERNOON SESSION DETAILS** | | | |
| **Image of session** | | | |
| **Report**  **Encapsulation**  **Encapsulation is to ensure that implementation details are not visible to users. The variables of one class will be hidden from the other classes, accessible only through the methods of the current class. This is called data hiding.**  **To achieve encapsulation in Java, declare the class' variables as private and provide public setter and getter methods to modify and view the variables' values.**  **Eg:**  **class BankAccount {**  **private double balance=0;**  **public void deposit(double x) {**  **if(x > 0) {**  **balance += x;**  **}**  **}**  **}**  **Inheritance**  **Inheritance is the process that enables one class to acquire the properties (methods and variables) of another. With inheritance, the information is placed in a more manageable, hierarchical order.**  **The class inheriting the properties of another is the subclass. The class whose properties are inherited is the superclass.**  **class Animal {**  **protected int legs;**  **public void eat() {**  **System.out.println("Animal eats");**  **}**  **}**  **class Dog extends Animal {**  **Dog() {**  **legs = 4;**  **}**  **}**  **class MyClass {**  **public static void main(String[ ] args) {**  **Dog d = new Dog();**  **d.eat();**  **}**  **}**  **Polymorphism**  **Polymorphism, which refers to the idea of "having many forms", occurs when there is a hierarchy of classes related to each other through inheritance.**  **A call to a member method will cause a different implementation to be executed, depending on the type of the object invoking the method.**  **Eg:**  **class Animal {**  **public void makeSound() {**  **System.out.println("Grr...");**  **}**  **}**  **class Cat extends Animal {**  **public void makeSound() {**  **System.out.println("Meow");**  **}**  **}**  **class Dog extends Animal {**  **public void makeSound() {**  **System.out.println("Woof");**  **}**  **}**  **class Program {**  **public static void main(String args[ ]) {**  **Animal a = new Dog();**  **Animal b = new Cat();**    **a.makeSound();**  **b.makeSound();**  **}**  **}**  **Overriding and Overloading**  **Subclass can define a behavior that's specific to the subclass type, meaning that a subclass can implement a parent class method based on its requirement.**  **This feature is known as method overriding.**  **- Should have the same return type and arguments**  **- The access level cannot be more restrictive than the overridden method's access level.**  **- A method declared final or static cannot be overridden**  **- If a method cannot be inherited, it cannot be overridden**  **- Constructors cannot be overridden**  **When methods have the same name, but different parameters, it is known as method overloading.**  **This can be very useful when you need the same method functionality for different types of parameters.**  **Eg:**  **class Program {**  **static double max(double a, double b) {**  **if(a > b) {**  **return a;**  **}**  **else {**  **return b;**  **}**  **}**  **static int max(int a, int b) {**  **if(a > b) {**  **return a;**  **}**  **else {**  **return b;**  **}**  **}**  **public static void main(String[] args) {**  **System.out.println(max(8, 17));**  **System.out.println(max(3.14, 7.68));**  **}**  **}**  **Abstracts Class**  **Data abstraction provides the outside world with only essential information, in a process of representing essential features without including implementation details.**  **A good real-world example is a book. When you hear the term book, you don't know the exact specifics, such as the page count, the color, or the size, but you understand the idea, or abstraction, of a book.**  **The concept of abstraction is that we focus on essential qualities, rather than the specific characteristics of one particular.**  **Eg:**  **abstract class Animal {**  **int legs = 0;**  **abstract void makeSound();**  **}**  **class Cat extends Animal {**  **public void makeSound() {**  **System.out.println("Meow");**  **}**  **}**  **public class Program {**  **public static void main(String[] args) {**  **Cat c = new Cat();**  **c.makeSound();**  **}**  **}** | | | |