**РУСЕНСКИ УНИВЕРСИТЕТ “АНГЕЛ КЪНЧЕВ”**

КУРСОВА РАБОТА  
ПО ПРОГРАМНИ ЕЗИЦИ

Студент:

Факултетен номер:

Група:

Специалност:

Дата: Изготвил:  
 Проверил:

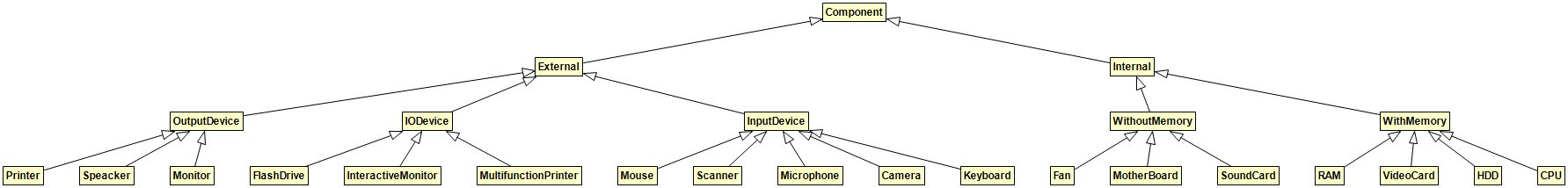
# Задание.

Да се състави йерархия от класове, описващи **компютърни компоненти** и програма на Java с графичен потреителски интерфейс, включваща следните функционалности:

* въвеждане на данни за обекти от тези класове;
* съхранение на данните;

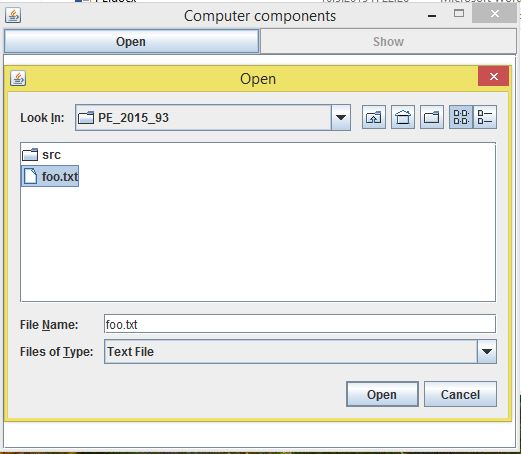
Данните за обектите да се попълват от **текстов файл, като на всеки ред във файла е стойността на едно поле, а между обектите се използва подходящ разделител** и да се съхраняват в **масив**. Изборът на атрибути и методи е на студента, но броят, типовете им и разположението име в йерархията влияят на оценката.

# Клас диаграми.

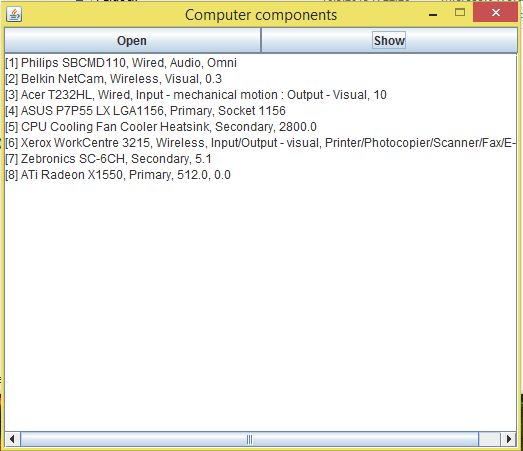


Фигура 1 Клас диаграма.

# Тестови примери



Фигура 2 Избор на файл



Фигура 3 Визуализиране на данните

# Листинг.

MAIN.JAVA

package pc;

public class Main {

private static GUI gui;

public static void main(String[] args) {

setGui(new GUI("Computer components"));

}

public static GUI getGui() {

return gui;

}

public static void setGui(GUI gui) {

Main.gui = gui;

}

}

GUI.JAVA

package pc;

import java.awt.BorderLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.FileNotFoundException;

import javax.swing.JButton;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTextArea;

import javax.swing.filechooser.FileFilter;

import javax.swing.filechooser.FileNameExtensionFilter;

import components.pc.\*;

public class GUI implements ActionListener {

public static final String OBJECT\_DELIMITER = "\*#\*#\*#\*";

private File file;

private JTextArea taContent;

private JButton btnOpen;

private JButton btnShow;

private JFileChooser fc;

private JFrame frame;

private Component[] arr;

public GUI(String title) {

frame = new JFrame(title);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.getContentPane().setLayout(new BorderLayout());

fc = new JFileChooser();

FileFilter fileFilter = new FileNameExtensionFilter("Text File", "txt");

fc.setFileFilter(fileFilter);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

JPanel pnlButtons = new JPanel(new GridLayout(0, 2));

btnOpen = new JButton("Open");

btnOpen.addActionListener(this);

pnlButtons.add(btnOpen);

btnShow = new JButton("Show");

btnShow.addActionListener(this);

btnShow.setEnabled(false);

pnlButtons.add(btnShow);

frame.getContentPane().add(pnlButtons, BorderLayout.PAGE\_START);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

JPanel pnlContent = new JPanel(new BorderLayout());

taContent = new JTextArea();

JScrollPane scroll = new JScrollPane (taContent,

JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED, JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED);

pnlContent.add(scroll, BorderLayout.CENTER);

frame.getContentPane().add(pnlContent, BorderLayout.CENTER);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

frame.pack();

frame.setSize(520, 450);

frame.setResizable(false);

frame.setLocationRelativeTo(null);

frame.setVisible(true);

}

private void printObejctsIntoTextArea() {

taContent.setText("");

int count = 0;

for (Component f : arr) {

taContent.append("[" + ++count + "] ");

taContent.append(f.toString());

taContent.append("\r\n");

}

}

@Override

public void actionPerformed(ActionEvent e) {

if (e.getSource() == btnOpen) {

int returnVal = fc.showOpenDialog(frame);

if (returnVal == JFileChooser.APPROVE\_OPTION) {

file = fc.getSelectedFile();

btnShow.setEnabled(true);

readFile();

} else if (file == null) {

btnShow.setEnabled(false);

}

} else if (e.getSource() == btnShow) {

printObejctsIntoTextArea();

}

}

private void readFile() {

if (file != null && !file.getName().isEmpty()) {

java.util.Scanner fileScanner;

try {

fileScanner = new java.util.Scanner(file);

int objCounter = 0;

while (fileScanner.hasNextLine()) {

if (fileScanner.nextLine().compareTo(OBJECT\_DELIMITER) == 0) {

objCounter++;

}

}

if (objCounter > 0) {

arr = new Component[objCounter];

objCounter = -1;

fileScanner = new java.util.Scanner(file);

boolean newObject = false;

String line;

int c = 0;

while (fileScanner.hasNextLine()) {

line = fileScanner.nextLine();

if (line.compareTo(OBJECT\_DELIMITER) == 0) {

newObject = true;

objCounter++;

continue;

}

if (newObject == true) {

c = 0;

newObject = false;

switch (line) {

case PCComponents.CAMERA:

Camera camera = new Camera();

while (fileScanner.hasNext()) {

if (c == 0) { camera.setName(fileScanner.nextLine()); }

else if (c == 1) { camera.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { camera.setModalityOfInput(fileScanner.nextLine()); }

else if (c == 3) { camera.setResolution(Float.parseFloat(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = camera;

break;

case PCComponents.COMPONENT:

Component component = new Component();

while (fileScanner.hasNext()) {

if (c == 0) { component.setName(fileScanner.nextLine()); }

else break;

c++;

}

arr[objCounter] = component;

break;

case PCComponents.CPU:

CPU cpu = new CPU();

while (fileScanner.hasNext()) {

if (c == 0) { cpu.setName(fileScanner.nextLine()); }

else if (c == 1) { cpu.setPrimary(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { cpu.setMemoryCapacity(Float.parseFloat(fileScanner.nextLine())); }

else if (c == 3) { cpu.setCacheLevels(Integer.parseInt(fileScanner.nextLine())); }

else if (c == 4) { cpu.setNumberOfCores(Integer.parseInt(fileScanner.nextLine())); }

else if (c == 5) { cpu.setClockSpeed(Float.parseFloat(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = cpu;

break;

case PCComponents.EXTERNAL:

External external = new External();

while (fileScanner.hasNext()) {

if (c == 0) { external.setName(fileScanner.nextLine()); }

else if (c == 1) { external.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = external;

break;

case PCComponents.FAN:

Fan fan = new Fan();

while (fileScanner.hasNext()) {

if (c == 0) { fan.setName(fileScanner.nextLine()); }

else if (c == 1) { fan.setPrimary(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { fan.setRotationalSpeed(Float.parseFloat(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = fan;

break;

case PCComponents.FLASH\_DRIVE:

FlashDrive fd = new FlashDrive();

while (fileScanner.hasNext()) {

if (c == 0) { fd.setName(fileScanner.nextLine()); }

else if (c == 1) { fd.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { fd.setModalityOfIO(fileScanner.nextLine()); }

else if (c == 3) { fd.setCapacity(Float.parseFloat(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = fd;

break;

case PCComponents.HDD:

HDD hdd = new HDD();

while (fileScanner.hasNext()) {

if (c == 0) { hdd.setName(fileScanner.nextLine()); }

else if (c == 1) { hdd.setPrimary(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { hdd.setMemoryCapacity(Float.parseFloat(fileScanner.nextLine())); }

else if (c == 3) { hdd.setLatency(Float.parseFloat(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = hdd;

break;

case PCComponents.INPUT\_DEVICE:

InputDevice id = new InputDevice();

while (fileScanner.hasNext()) {

if (c == 0) { id.setName(fileScanner.nextLine()); }

else if (c == 1) { id.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { id.setModalityOfInput(fileScanner.nextLine()); }

else break;

c++;

}

arr[objCounter] = id;

break;

case PCComponents.INTERACTIVE\_MONITOR:

InteractiveMonitor im = new InteractiveMonitor();

while (fileScanner.hasNext()) {

if (c == 0) { im.setName(fileScanner.nextLine()); }

else if (c == 1) { im.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { im.setModalityOfIO(fileScanner.nextLine()); }

else if (c == 3) { im.setTouchPoints(Integer.parseInt(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = im;

break;

case PCComponents.INTERNAL:

Internal internal = new Internal();

while (fileScanner.hasNext()) {

if (c == 0) { internal.setName(fileScanner.nextLine()); }

else if (c == 1) { internal.setPrimary(Boolean.parseBoolean(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = internal;

break;

case PCComponents.IO\_DEVICE:

IODevice iod = new IODevice();

while (fileScanner.hasNext()) {

if (c == 0) { iod.setName(fileScanner.nextLine()); }

else if (c == 1) { iod.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { iod.setModalityOfIO(fileScanner.nextLine()); }

else break;

c++;

}

arr[objCounter] = iod;

break;

case PCComponents.KEYBOARD:

Keyboard kb = new Keyboard();

while (fileScanner.hasNext()) {

if (c == 0) { kb.setName(fileScanner.nextLine()); }

else if (c == 1) { kb.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { kb.setModalityOfInput(fileScanner.nextLine()); }

else if (c == 3) { kb.setCountryLayout(fileScanner.nextLine()); }

else if (c == 4) { kb.setIntegratedTrackball(Boolean.parseBoolean(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = kb;

break;

case PCComponents.MICROPHONE:

Microphone mp = new Microphone();

while (fileScanner.hasNext()) {

if (c == 0) { mp.setName(fileScanner.nextLine()); }

else if (c == 1) { mp.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { mp.setModalityOfInput(fileScanner.nextLine()); }

else if (c == 3) { mp.setPickUpPattern(fileScanner.nextLine()); }

else break;

c++;

}

arr[objCounter] = mp;

break;

case PCComponents.MONITOR:

Monitor mon = new Monitor();

while (fileScanner.hasNext()) {

if (c == 0) { mon.setName(fileScanner.nextLine()); }

else if (c == 1) { mon.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { mon.setNumberOfColors(Integer.parseInt(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = mon;

break;

case PCComponents.MOTHERBOARD:

MotherBoard mb = new MotherBoard();

while (fileScanner.hasNext()) {

if (c == 0) { mb.setName(fileScanner.nextLine()); }

else if (c == 1) { mb.setPrimary(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { mb.setCpuSocket(fileScanner.nextLine()); }

else break;

c++;

}

arr[objCounter] = mb;

break;

case PCComponents.MOUSE:

Mouse mo = new Mouse();

while (fileScanner.hasNext()) {

if (c == 0) { mo.setName(fileScanner.nextLine()); }

else if (c == 1) { mo.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { mo.setModalityOfInput(fileScanner.nextLine()); }

else if (c == 3) { mo.setSensitivity(Float.parseFloat(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = mo;

break;

case PCComponents.MULTIFUNCTION\_PRINTER:

MultifunctionPrinter mup = new MultifunctionPrinter();

while (fileScanner.hasNext()) {

if (c == 0) { mup.setName(fileScanner.nextLine()); }

else if (c == 1) { mup.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { mup.setModalityOfIO(fileScanner.nextLine()); }

else if (c == 3) { mup.setIncludedDevices(fileScanner.nextLine()); }

else break;

c++;

}

arr[objCounter] = mup;

break;

case PCComponents.OUTPUT\_DEVICE:

OutputDevice od = new OutputDevice();

while (fileScanner.hasNext()) {

if (c == 0) { od.setName(fileScanner.nextLine()); }

else if (c == 1) { od.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { od.setModalityOfOutput(fileScanner.nextLine()); }

else break;

c++;

}

arr[objCounter] = od;

break;

case PCComponents.PRINTER:

Printer p = new Printer();

while (fileScanner.hasNext()) {

if (c == 0) { p.setName(fileScanner.nextLine()); }

else if (c == 1) { p.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { p.setModalityOfOutput(fileScanner.nextLine()); }

else if (c == 3) { p.setMonochrome(Boolean.parseBoolean(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = p;

break;

case PCComponents.RAM:

RAM ram = new RAM();

while (fileScanner.hasNext()) {

if (c == 0) { ram.setName(fileScanner.nextLine()); }

else if (c == 1) { ram.setPrimary(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { ram.setMemoryCapacity(Float.parseFloat(fileScanner.nextLine())); }

else if (c == 3) { ram.setFrequency(Float.parseFloat(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = ram;

break;

case PCComponents.SCANNER:

Scanner sc = new Scanner();

while (fileScanner.hasNext()) {

if (c == 0) { sc.setName(fileScanner.nextLine()); }

else if (c == 1) { sc.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { sc.setModalityOfInput(fileScanner.nextLine()); }

else if (c == 3) { sc.setResolution(Float.parseFloat(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = sc;

break;

case PCComponents.SOUND\_CARD:

SoundCard soc = new SoundCard();

while (fileScanner.hasNext()) {

if (c == 0) { soc.setName(fileScanner.nextLine()); }

else if (c == 1) { soc.setPrimary(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { soc.setChannel(fileScanner.nextLine()); }

else break;

c++;

}

arr[objCounter] = soc;

break;

case PCComponents.SPEACKER:

Speacker sp = new Speacker();

while (fileScanner.hasNext()) {

if (c == 0) { sp.setName(fileScanner.nextLine()); }

else if (c == 1) { sp.setWired(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { sp.setModalityOfOutput(fileScanner.nextLine()); }

else break;

c++;

}

arr[objCounter] = sp;

break;

case PCComponents.VIDEO\_CARD:

VideoCard vc = new VideoCard();

while (fileScanner.hasNext()) {

if (c == 0) { vc.setName(fileScanner.nextLine()); }

else if (c == 1) { vc.setPrimary(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { vc.setMemoryCapacity(Float.parseFloat(fileScanner.nextLine())); }

else if (c == 2) { vc.setBandwidth(Float.parseFloat(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = vc;

break;

case PCComponents.WITH\_MEMORY:

WithMemory wm = new WithMemory();

while (fileScanner.hasNext()) {

if (c == 0) { wm.setName(fileScanner.nextLine()); }

else if (c == 1) { wm.setPrimary(Boolean.parseBoolean(fileScanner.nextLine())); }

else if (c == 2) { wm.setMemoryCapacity(Float.parseFloat(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = wm;

break;

case PCComponents.WITHOUT\_MEMORY:

WithoutMemory wom = new WithoutMemory();

while (fileScanner.hasNext()) {

if (c == 0) { wom.setName(fileScanner.nextLine()); }

else if (c == 1) { wom.setPrimary(Boolean.parseBoolean(fileScanner.nextLine())); }

else break;

c++;

}

arr[objCounter] = wom;

break;

default:

taContent.setText("Error!");

System.out.println("Error!");

break;

}

}

}

}

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

}

}

HDD.JAVA

package components.pc;

public class HDD extends WithMemory {

private float latency;

public float getLatency() {

return latency;

}

public void setLatency(float latency) {

this.latency = latency;

}

public HDD() {

}

public HDD(String name, boolean primary, float memoryCapacity,

float speed) {

super(name, primary, memoryCapacity);

this.latency = speed;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.latency);

return sb.toString();

}

}

CAMERA.JAVA

package components.pc;

public class Camera extends InputDevice {

private float resolution;

public float getResolution() {

return resolution;

}

public void setResolution(float resolution) {

this.resolution = resolution;

}

public Camera() {

}

public Camera(String name, boolean wired, String modalityOfInput,

float resolution) {

super(name, wired, modalityOfInput);

this.resolution = resolution;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.resolution);

return sb.toString();

}

}

COMPONENT.JAVA

package components.pc;

public class Component {

private String name;

public Component() {

}

public Component(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return name;

}

}

CPU.JAVA

package components.pc;

public class CPU extends WithMemory {

private int cacheLevels;

private int numberOfCores;

private float clockSpeed;

public int getCacheLevels() {

return cacheLevels;

}

public void setCacheLevels(int cacheLevels) {

this.cacheLevels = cacheLevels;

}

public int getNumberOfCores() {

return numberOfCores;

}

public void setNumberOfCores(int number) {

this.numberOfCores = number;

}

public float getClockSpeed() {

return clockSpeed;

}

public void setClockSpeed(float clockSpeed) {

this.clockSpeed = clockSpeed;

}

public CPU() {

}

public CPU(String name, boolean primary, float memoryCapacity,

int cacheLevels, int coreNumber, float clockSpeed) {

super(name, primary, memoryCapacity);

this.cacheLevels = cacheLevels;

this.numberOfCores = coreNumber;

this.clockSpeed = clockSpeed;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.cacheLevels);

sb.append(", ");

sb.append(this.numberOfCores);

sb.append(", ");

sb.append(this.clockSpeed);

return sb.toString();

}

}

EXTERNAL.JAVA

package components.pc;

public class External extends Component {

private boolean wired;

public boolean isWired() {

return wired;

}

public void setWired(boolean wired) {

this.wired = wired;

}

public External() {

}

public External(String name, boolean wired) {

super(name);

this.wired = wired;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

if (this.wired) {

sb.append("Wired");

}

else {

sb.append("Wireless");

}

return sb.toString();

}

}

FAN.JAVA

package components.pc;

public class Fan extends WithoutMemory {

private float rotationalSpeed;

public float getRotationalSpeed() {

return rotationalSpeed;

}

public void setRotationalSpeed(float rotationalSpeed) {

this.rotationalSpeed = rotationalSpeed;

}

public Fan() {

}

public Fan(String name, boolean primary, float rotationalSpeed) {

super(name, primary);

this.rotationalSpeed = rotationalSpeed;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.rotationalSpeed);

return sb.toString();

}

}

FLASHDRIVE.JAVA

package components.pc;

public class FlashDrive extends IODevice {

private float capacity;

public float getCapacity() {

return capacity;

}

public void setCapacity(float capacity) {

this.capacity = capacity;

}

public FlashDrive() {

}

public FlashDrive(String name, boolean wired,

String modalityOfIO, float capacity) {

super(name, wired, modalityOfIO);

this.capacity = capacity;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.capacity);

return sb.toString();

}

}

INPUTDEVICE.JAVA

package components.pc;

public class InputDevice extends External {

private String modalityOfInput; // mechanical motion, audio, visual, etc.

public String getModalityOfInput() {

return modalityOfInput;

}

public void setModalityOfInput(String modalityOfInput) {

this.modalityOfInput = modalityOfInput;

}

public InputDevice() {

}

public InputDevice(String name, boolean wired, String modalityOfInput) {

super(name, wired);

this.modalityOfInput = modalityOfInput;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.modalityOfInput);

return sb.toString();

}

}

INTERACTIVEMONITOR.JAVA

package components.pc;

public class InteractiveMonitor extends IODevice {

private int touchPoints;

public int getTouchPoints() {

return touchPoints;

}

public void setTouchPoints(int touchPoints) {

this.touchPoints = touchPoints;

}

public InteractiveMonitor() {

}

public InteractiveMonitor(String name, boolean wired,

String modalityOfIO, int touchPoints) {

super(name, wired, modalityOfIO);

this.touchPoints = touchPoints;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.touchPoints);

return sb.toString();

}

}

INTERNAL.JAVA

package components.pc;

public class Internal extends Component {

private boolean primary;

public boolean isPrimary() {

return primary;

}

public void setPrimary(boolean primary) {

this.primary = primary;

}

public Internal() {

}

public Internal(String name, boolean primary) {

super(name);

this.primary = primary;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

if (this.primary) {

sb.append("Primary");

}

else {

sb.append("Secondary");

}

return sb.toString();

}

}

IODEVICE.JAVA

package components.pc;

public class IODevice extends External {

private String modalityOfIO;

public String getModalityOfIO() {

return modalityOfIO;

}

public void setModalityOfIO(String modalityOfIO) {

this.modalityOfIO = modalityOfIO;

}

public IODevice() {

}

public IODevice(String name, boolean wired, String modalityOfIO) {

super(name, wired);

this.modalityOfIO = modalityOfIO;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.modalityOfIO);

return sb.toString();

}

}

KEYBOARD.JAVA

package components.pc;

public class Keyboard extends InputDevice {

private String countryLayout;

private boolean integratedTrackball;

public String getCountryLayout() {

return countryLayout;

}

public void setCountryLayout(String countryLayout) {

this.countryLayout = countryLayout;

}

public boolean hasIntegratedTrackball() {

return integratedTrackball;

}

public void setIntegratedTrackball(boolean integratedTrackball) {

this.integratedTrackball = integratedTrackball;

}

public Keyboard() {

}

public Keyboard(String name, boolean wired, String modalityOfInput,

String countryLayout, boolean integratedTrackball) {

super(name, wired, modalityOfInput);

this.countryLayout = countryLayout;

this.integratedTrackball = integratedTrackball;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.countryLayout);

sb.append(", ");

if (this.integratedTrackball) {

sb.append("Has integrated trackball");

}

else {

sb.append("Doesn't have integrated trackball");

}

return sb.toString();

}

}

MICROPHONE.JAVA

package components.pc;

public class Microphone extends InputDevice {

private String pickUpPattern; /\* omni, bi-directional, cardioid,

hyper-cardioid, shotgun \*/

public String getPickUpPattern() {

return pickUpPattern;

}

public void setPickUpPattern(String pickUpPattern) {

this.pickUpPattern = pickUpPattern;

}

public Microphone() {

}

public Microphone(String name, boolean wired, String modalityOfInput,

String pickUpPattern) {

super(name, wired, modalityOfInput);

this.pickUpPattern = pickUpPattern;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.pickUpPattern);

return sb.toString();

}

}

MONITOR.JAVA

package components.pc;

public class Monitor extends OutputDevice {

private int numberOfColors;

public int getNumberOfColors() {

return numberOfColors;

}

public void setNumberOfColors(int numberOfColors) {

this.numberOfColors = numberOfColors;

}

public Monitor() {

}

public Monitor(String name, boolean wired, String modalityOfOuput,

int numberOfColors) {

super(name, wired, modalityOfOuput);

this.numberOfColors = numberOfColors;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.numberOfColors);

return sb.toString();

}

}

MOTHERBOARD.JAVA

package components.pc;

public class MotherBoard extends WithoutMemory {

private String cpuSocket;

public String getCpuSocket() {

return cpuSocket;

}

public void setCpuSocket(String cpuSocket) {

this.cpuSocket = cpuSocket;

}

public MotherBoard() {

}

public MotherBoard(String name, boolean primary, String cpuSocket) {

super(name, primary);

this.cpuSocket = cpuSocket;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.cpuSocket);

return sb.toString();

}

}

MOUSE.JAVA

package components.pc;

public class Mouse extends InputDevice {

private float sensitivity;

public float getSensitivity() {

return sensitivity;

}

public void setSensitivity(float sensitivity) {

this.sensitivity = sensitivity;

}

public Mouse() {

}

public Mouse(String name, boolean wired, String modalityOfInput,

float sensitivity) {

super(name, wired, modalityOfInput);

this.sensitivity = sensitivity;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.sensitivity);

return sb.toString();

}

}

MULTIFUNCTIONPRINTER.JAVA

package components.pc;

public class MultifunctionPrinter extends IODevice {

private String includedDevices;

public String getIncludedDevices() {

return includedDevices;

}

public void setIncludedDevices(String includedDevices) {

this.includedDevices = includedDevices;

}

public MultifunctionPrinter() {

}

public MultifunctionPrinter(String name, boolean wired,

String modalityOfIO, String includedDevices) {

super(name, wired, modalityOfIO);

this.includedDevices = includedDevices;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.includedDevices);

return sb.toString();

}

}

OUTPUTDEVICE.JAVA

package components.pc;

public class OutputDevice extends External {

private String modalityOfOutput; // visual, audio, tactile, etc.

public String getModalityOfOutput() {

return modalityOfOutput;

}

public void setModalityOfOutput(String modalityOfOutput) {

this.modalityOfOutput = modalityOfOutput;

}

public OutputDevice() {

}

public OutputDevice(String name, boolean wired, String modalityOfOutput) {

super(name, wired);

this.modalityOfOutput = modalityOfOutput;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.modalityOfOutput);

return sb.toString();

}

}

PCCOMPONENTS.JAVA

package pc;

public class PCComponents {

static final String CAMERA = "1";

static final String COMPONENT = "2";

static final String CPU = "3";

static final String EXTERNAL = "4";

static final String FAN = "5";

static final String FLASH\_DRIVE = "6";

static final String HDD = "7";

static final String INPUT\_DEVICE = "8";

static final String INTERACTIVE\_MONITOR = "9";

static final String INTERNAL = "10";

static final String IO\_DEVICE = "11";

static final String KEYBOARD = "12";

static final String MICROPHONE = "13";

static final String MONITOR = "14";

static final String MOTHERBOARD = "15";

static final String MOUSE = "16";

static final String MULTIFUNCTION\_PRINTER = "17";

static final String OUTPUT\_DEVICE = "18";

static final String PRINTER = "19";

static final String RAM = "20";

static final String SCANNER = "21";

static final String SOUND\_CARD = "22";

static final String SPEACKER = "23";

static final String VIDEO\_CARD = "24";

static final String WITH\_MEMORY = "25";

static final String WITHOUT\_MEMORY = "26";

}

PRINTER.JAVA

package components.pc;

public class Printer extends OutputDevice {

private boolean monochrome;

public boolean isMonochrome() {

return monochrome;

}

public void setMonochrome(boolean monochrome) {

this.monochrome = monochrome;

}

public Printer() {

}

public Printer(String name, boolean wired, String modalityOfOuput,

boolean monochrome) {

super(name, wired, modalityOfOuput);

this.monochrome = monochrome;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

if (this.monochrome) {

sb.append("Monochrome");

}

else {

sb.append("Not monochrome");

}

return sb.toString();

}

}

RAM.JAVA

package components.pc;

public class RAM extends WithMemory {

private float frequency;

public float getFrequency() {

return frequency;

}

public void setFrequency(float frequency) {

this.frequency = frequency;

}

public RAM() {

}

public RAM(String name, boolean primary, float memoryCapacity,

float frequency) {

super(name, primary, memoryCapacity);

this.frequency = frequency;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.frequency);

return sb.toString();

}

}

SCANNER.JAVA

package components.pc;

public class Scanner extends InputDevice {

private float resolution;

public float getResolution() {

return resolution;

}

public void setResolution(float resolution) {

this.resolution = resolution;

}

public Scanner() {

}

public Scanner(String name, boolean wired, String modalityOfInput,

float resolution) {

super(name, wired, modalityOfInput);

this.resolution = resolution;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.resolution);

return sb.toString();

}

}

SOUNDCARD.JAVA

package components.pc;

public class SoundCard extends WithoutMemory {

private String channel; // 2.1, 5.1, 7.1, etc.

public String getChannel() {

return channel;

}

public void setChannel(String channel) {

this.channel = channel;

}

public SoundCard() {

}

public SoundCard(String name, boolean primary, String channel) {

super(name, primary);

this.channel = channel;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.channel);

return sb.toString();

}

}

SPEACKER.JAVA

package components.pc;

public class Speacker extends OutputDevice {

private float power;

public float getPower() {

return power;

}

public void setPower(float power) {

this.power = power;

}

public Speacker() {

}

public Speacker(String name, boolean wired, String modalityOfOuput,

float power) {

super(name, wired, modalityOfOuput);

this.power = power;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.power);

return sb.toString();

}

}

VIDEOCARD.JAVA

package components.pc;

public class VideoCard extends WithMemory {

private float bandwidth;

public float getBandwidth() {

return bandwidth;

}

public void setBandwidth(float bandwidth) {

this.bandwidth = bandwidth;

}

public VideoCard() {

}

public VideoCard(String name, boolean primary, float memoryCapacity,

float bandwidth) {

super(name, primary, memoryCapacity);

this.bandwidth = bandwidth;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.bandwidth);

return sb.toString();

}

}

WITHMEMORY.JAVA

package components.pc;

public class WithMemory extends Internal {

private float memoryCapacity;

public float getMemoryCapacity() {

return memoryCapacity;

}

public void setMemoryCapacity(float memoryCapacity) {

this.memoryCapacity = memoryCapacity;

}

public WithMemory() {

}

public WithMemory(String name, boolean primary, float memoryCapacity) {

super(name, primary);

this.memoryCapacity = memoryCapacity;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(super.toString());

sb.append(", ");

sb.append(this.memoryCapacity);

return sb.toString();

}

}

WITHOUTMEMORY.JAVA

package components.pc;

public class WithoutMemory extends Internal {

public WithoutMemory() {

}

public WithoutMemory(String name, boolean primary) {

super(name, primary);

}

@Override

public String toString() {

return super.toString();

}

}