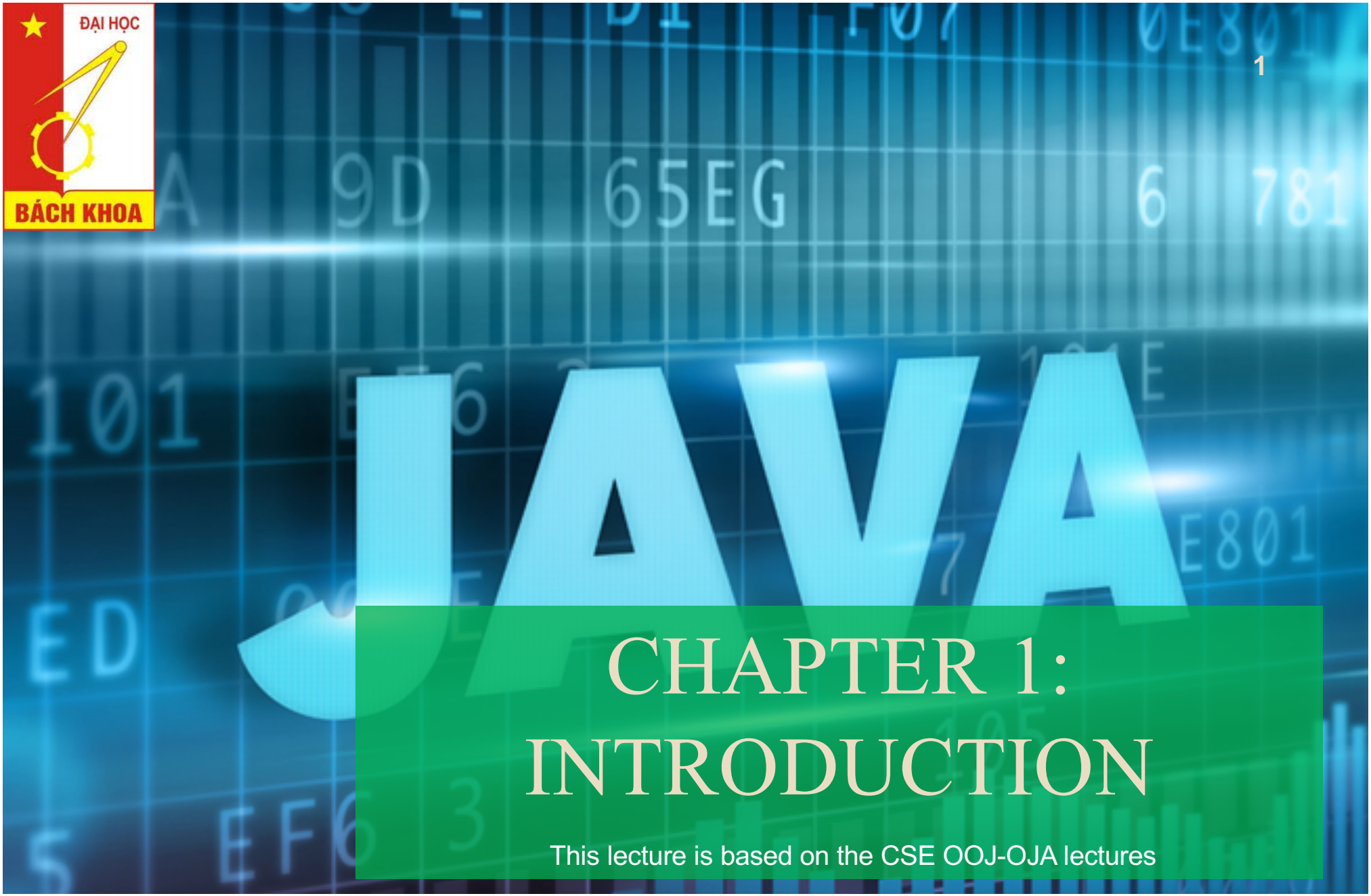


CHAPTER 1: INTRODUCTION

This lecture is based on the CSE OOJ-OJA lectures

Dr. Tran Hai Anh

Outline

1. Introduction to Java
2. Compiling and running Java programs
3. Object-oriented concepts

1. Introduction to Java

A small history of Java

- created by James Gosling from Sun Microsystems (Sun) in 1991.
- The target of Java: run a program on multiple OS.
- 1996: first publicly available version of Java (Java 1.0)
- 2006 Sun started to make Java available under the GNU General Public License (GPL).
- 2010: Sun Microsystems was acquired by the Oracle Corporation
- Oracle continues this project called *OpenJDK*.

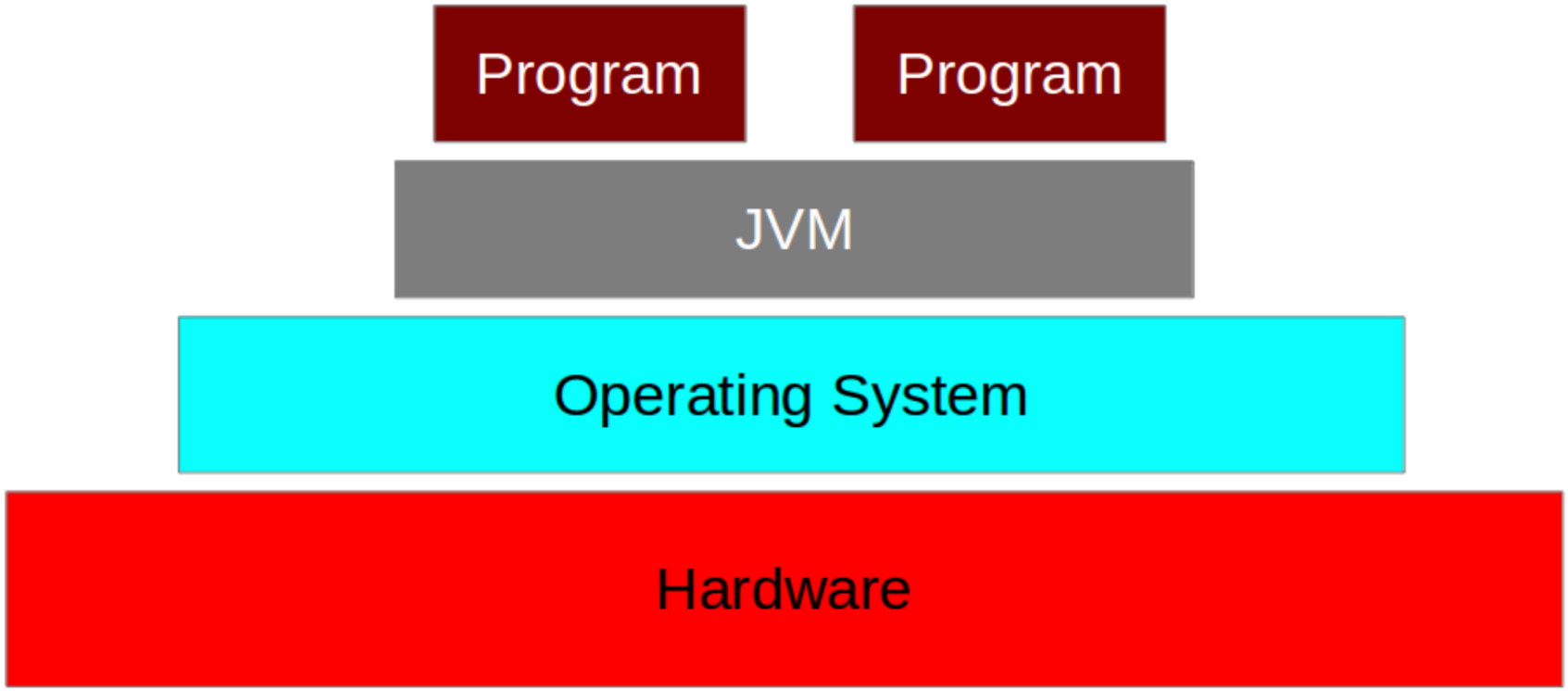
Properties

- Platform independent
- Object-orientated programming language
- Strongly-typed programming language
- Interpreted and compiled language
- Automatic memory management

Hello world Java program


```java
// a small Java program
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

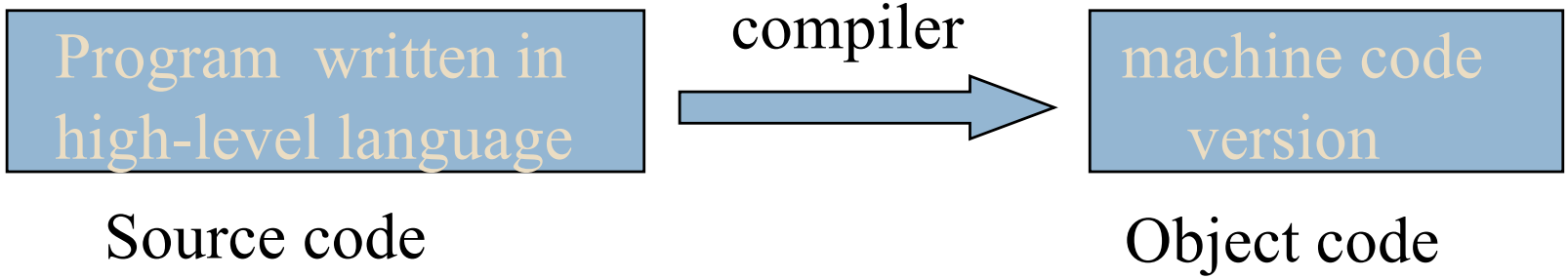
Java virtual machine



2. Compiling and running Java programs

Running high-level programs

- High-level language
  - Problem-oriented, must be translated to low-level
- Low-level language
  - What the machine actually executes
- Traditional compilation process

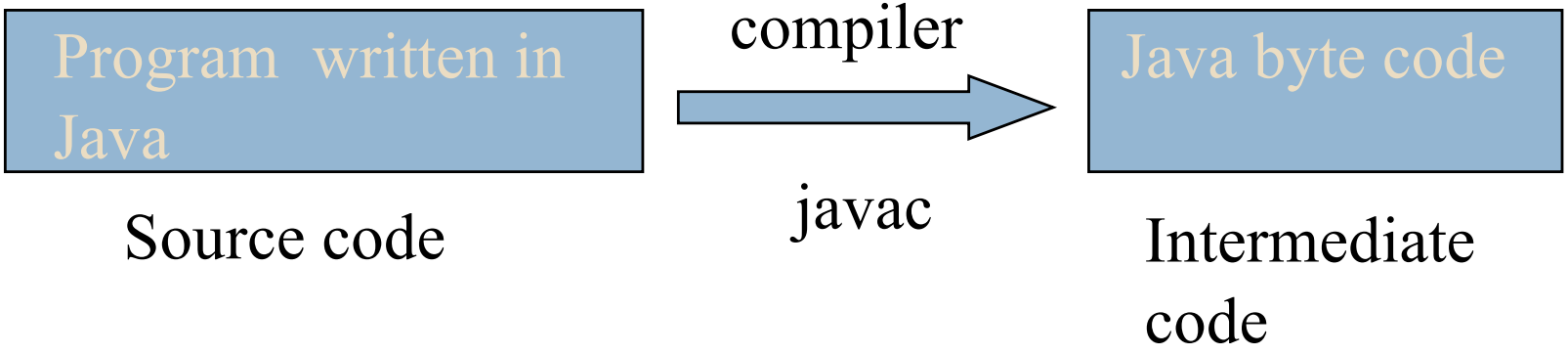


Byte code and JVM

- Programs written in high-level language are mostly translated into machine code, which is then directly executed by the CPU
- Java is an exception
- Java programs are translated into byte code, which is then executed by the Java Virtual Machine (JVM)
- The JVM is an interpreter program in machine code

Running Java programs

- Java compilation process



- Java execution process
  - Java byte code is read and executed via a Java byte code interpreter

Byte code and JVM

- The `javac` command converts the source programs into byte code
- Byte code files are those with the extension .class
- The `java` command causes the JVM to execute the byte code

(To increase execution speed, it is an option to convert the byte code into machine code)

Create, compile and run Java programs in Unix

- To create a file

  > vi <filename>

  Note: to create a Java program the filename must end in .java

- To compile a program

  > javac <filename>.java

- To run the program

  > java <classfilename>

14

3. Object-oriented concepts

3.1. Objects

Objects

- In the object-oriented paradigm everything is viewed as an object
- An object is an entity that has a state and behaviour
- The data that describe an object's state are called its attributes
- An object's behaviour is defined by the operations it can perform

pictures from www.amazon.com

Pop-up Toasters

Abstraction

- Each toaster has many attributes (colour, weight, height, price, etc.)
- In fact, it is not possible to list all the attributes of a toaster (or any object at all)
- We usually need to describe only some of its attributes - those that are of interest to us from a certain viewpoint
- Such a description is known as an abstraction

A view of the toasters

- Suppose we are now looking at the toasters from the viewpoint of *how they perform their toasting function*
- Then, from this viewpoint, we may describe them as shown in the next few slides

Toaster A

- 2 racks
- Each rack holds 2 slices of bread
- Darkness set at light
- Racks are up

Toaster B

- 4 racks
- Each rack holds 1 slice of bread
- Darkness is set at medium
- Racks are down

Toaster C

- 3 racks
- Each rack holds 1 slice of bread
- Darkness is set at medium-dark
- Racks are down

Toaster D

- 2 racks
- Each rack holds 1 slice of bread
- Darkness is set at medium
- Racks are up

Describing a Pop-up Toaster

- Attributes are pieces of data that describe the state of an object
- They have a value at any given time

State and attributes

- It is possible for two distinct objects to have the same state, i.e. the same values for their attributes
- E.g. if my brother and I both buy the same make and model of toaster and they are currently in the same state, they are still separate objects

Behaviour

- The behaviour of an object, like its attributes, is theoretically inexhaustible
- We usually only need to describe the behaviour of an object from a certain viewpoint
- In the object-oriented approach, we describe the behaviour of objects in terms of the operations they support

The behaviour of a Pop-up Toaster

- Is defined by the operations it supports
  - view darkness setting
  - change darkness setting
  - lower rack / start toasting
  - view rack status
  - stop toasting

3.2. Classes

Classes

- Classes describe a group of similar objects
- They form a template for the creation of instance objects
- Creating an instance object from a class template is called *instantiation*
- Classes determine what attributes an instance object of that type should have, though each instance object may have different values for each attribute
- Objects are examples of a class

Pop-up Toaster class

PopUpToaster
numberOfRacks rackSize darknessSetting rackStatus
getDarknessSetting( ) setDarknessSetting( ) getRackStatus( ) startToasting( ) stopToastingEarly( )

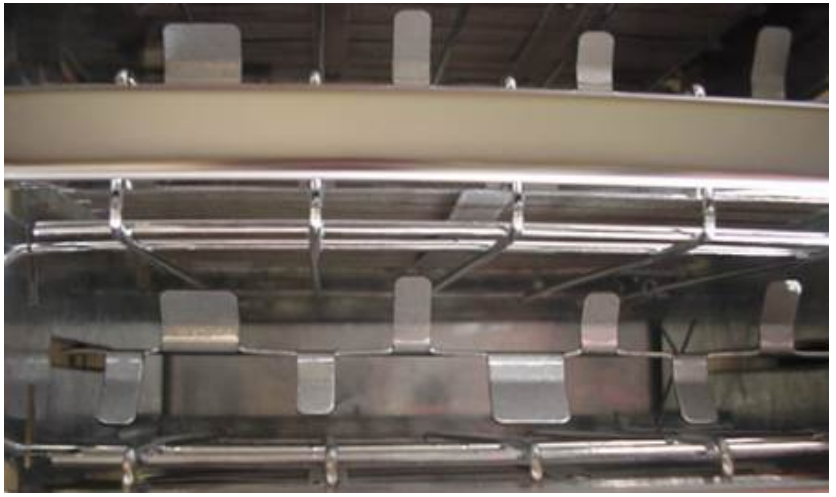
3.3. Information hiding

(also known as Encapsulation)

pictures and concepts from www.howstuffworks.com

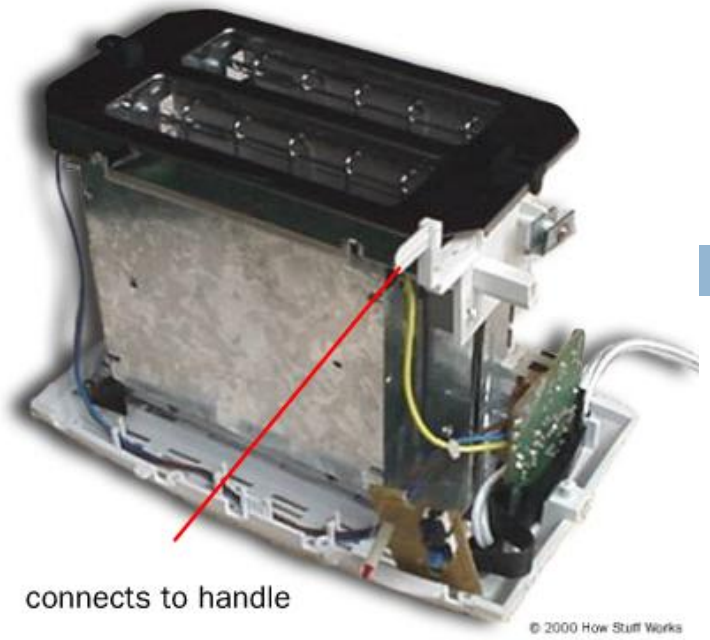
How toasters work

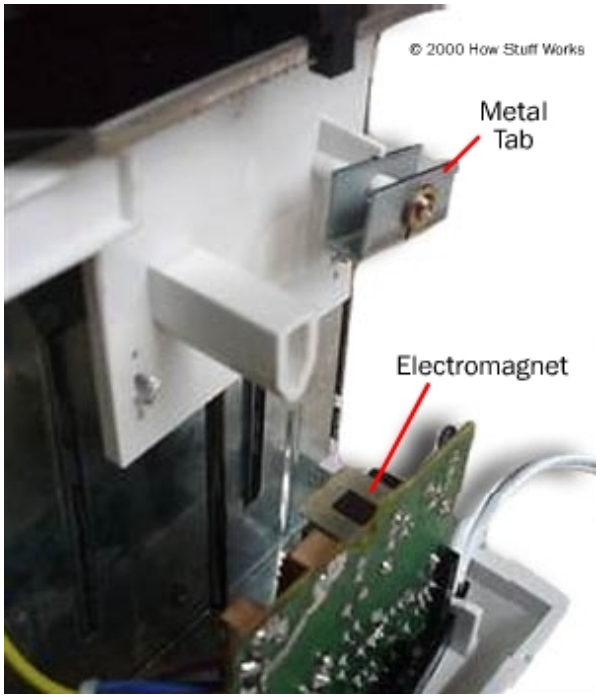
- Infrared radiation
- Nichrome wire wrapped across a mica sheet
- Spring-loaded tray
- Timer turns toaster off and releases rack
- Grates to centre bread



How toasters work

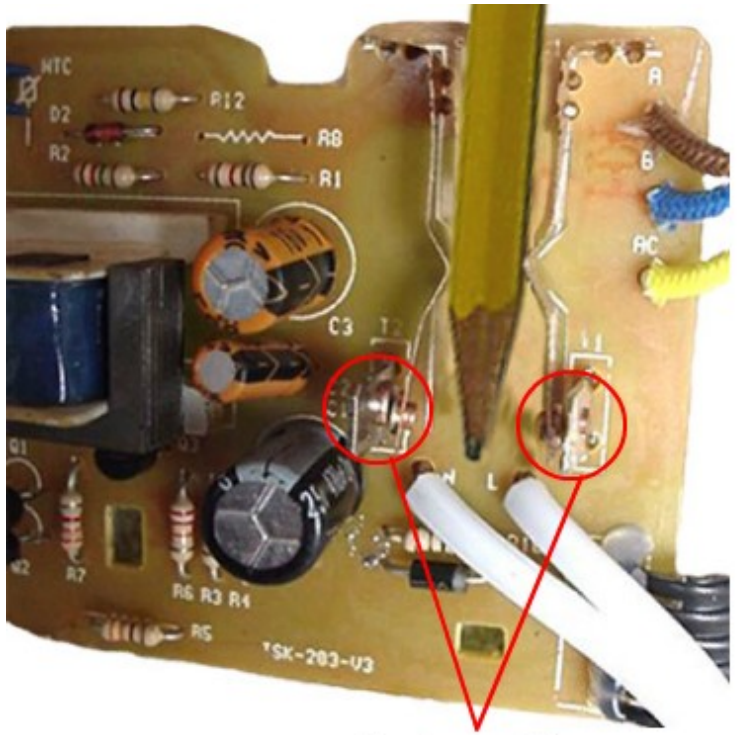
- Handle connected to rack to lower bread into toaster
- Electromagnet holds racks down





How toasters work

- Circuit board with contacts
- Variable resistor to control rate at which capacitor charges, which controls length of time before electromagnet is released (sets darkness)



How toasters work

- When you push down on the handle,
  - The plastic bar presses against the contacts and applies power to the circuit board
  - Power runs through the contacts to the nichrome wires to start toasting the bread
  - A circuit made up of transistors, resistors and capacitors, turns on and supplies power to the electromagnet
  - The electromagnet attracts the piece of metal on the handle, holding the bread in the toaster
  - The circuit acts as a timer. A capacitor charges and when it reaches a certain voltage it cuts off the power to the electromagnet. The spring pulls the two slices of bread up.
  - The plastic bar rises and cuts off power to the toaster.

How toasters work

- The darkness control is simply a variable resistor
  - Changing the resistance changes the rate at which the capacitor charges, and this controls how long the timer waits before releasing the electromagnet

Using a toaster

- To use a toaster, do you need to know how the toaster works?
- In fact most of the mechanism is hidden from view
- To use a toaster, we only need to be able to use its controls (operations)
- A pop-up toaster has few operations to control it
- We refer to the set of operations available to its user as its interface

Information hiding

- In the object-oriented paradigm, the details of an object that do not need to be known to use that object are hidden from view
- A user is only allowed to know about the details necessary to operate the object
- The set of operations visible to a user of an object is called the object's interface

Information hiding

- We may not want to allow access to the internals of an object for a number of reasons
  - Confusing and unnecessary for a user to know
  - Dangerous to the user of the object
  - Dangerous to the object
- The property of providing a limited interface and hiding the details is called information hiding or encapsulation

Message passing

- To request that an object performs one of its operations, a message must be sent to that object
- Message passing is the name given to the process of sending a message to an object to request the execution of one of its operations
- A message must have
  - A receiver (the object to which it is sent)
  - An operation selector (the name of the operation to be carried out)
- A message may also have
  - Associated data to work with

3.4. Operations and methods

Operations and methods

- An operation is the name of a task that can be carried out
- The way the operation works is called its method
- The method specifies how the operation is to be carried out
- The method is described by some algorithm (sequence of steps performed in doing the operation)
- Sometimes the terms operation and method are used interchangeably

Constructors

- When we create instance objects from a class template, we may want to initialise some of its values
- A constructor is a special operation that is performed when we create an instance of a class
- The constructor is generally used to give initial values to an object's attributes
- E.g. when we construct a toaster instance we may want to set the rack size and number of racks: this could be done in a constructor

Accessor and mutator methods

- By encapsulating attributes, we hide them from outside view
- Sometimes we need to allow other objects or users to find out or change the value of an attribute
- A method that simply allows the user to view the state of an attribute is called an accessor method
- E.g. getDarknessSetting( )

Accessor and mutator methods

- A method that simply allows the user to set the state of an attribute to a new value is called a mutator method
- E.g. setDarknessSetting( )

Class attributes and methods

- Instance attributes describe the state of an instance object (e.g. darknessSetting)
- Instance methods work in relation to a specific object (e.g. startToasting( ))
- Sometimes we want to store information about the class as a whole, such as totals or averages: these are called class attributes

Class attributes and methods

- For example we could store the total number of toaster instances, or the average rack size of all toaster instances
- Class methods work with a class as a whole and not on an individual instance (e.g. calculateAverageRackSize( ))

3.5. Inheritance

Inheritance

Toaster

PopUpToaster

RollerToaster

ToasterOven

Inheritance

- Is the way we define specialisation and generalisation among classes
- A superclass is more generalised than its subclasses
- A toaster is a more general form which includes pop-up toasters, roller toasters and toaster ovens

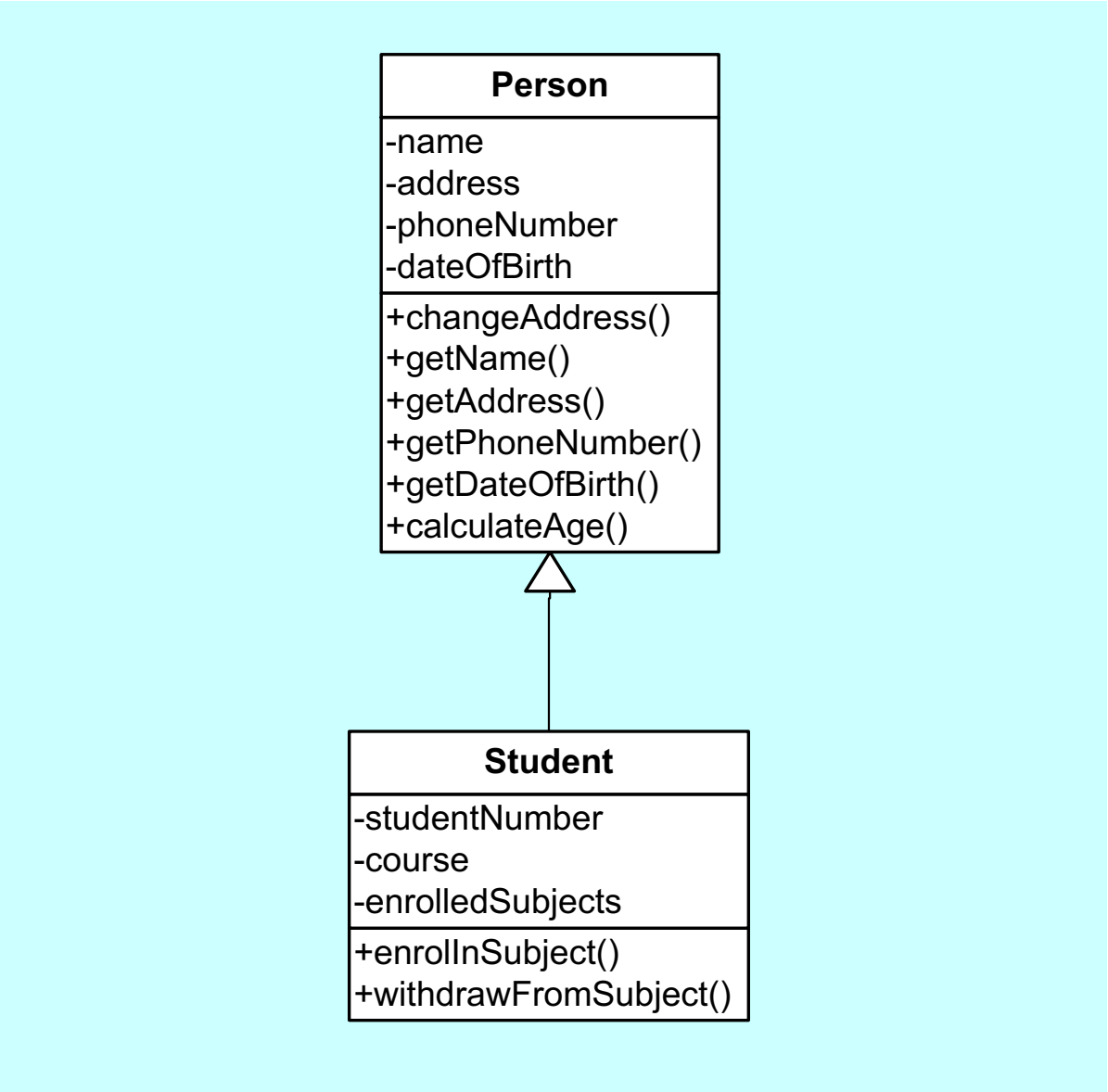
Inheritance

- Allows us to define properties (both attributes and operations) common to a number of classes once
- Allows us to define specialised classes which can access attributes and operations defined in a general class
- Allows us to refer to different types of objects collectively
  - E.g. we can refer to pop-up toasters and toaster ovens collectively as “toasters”

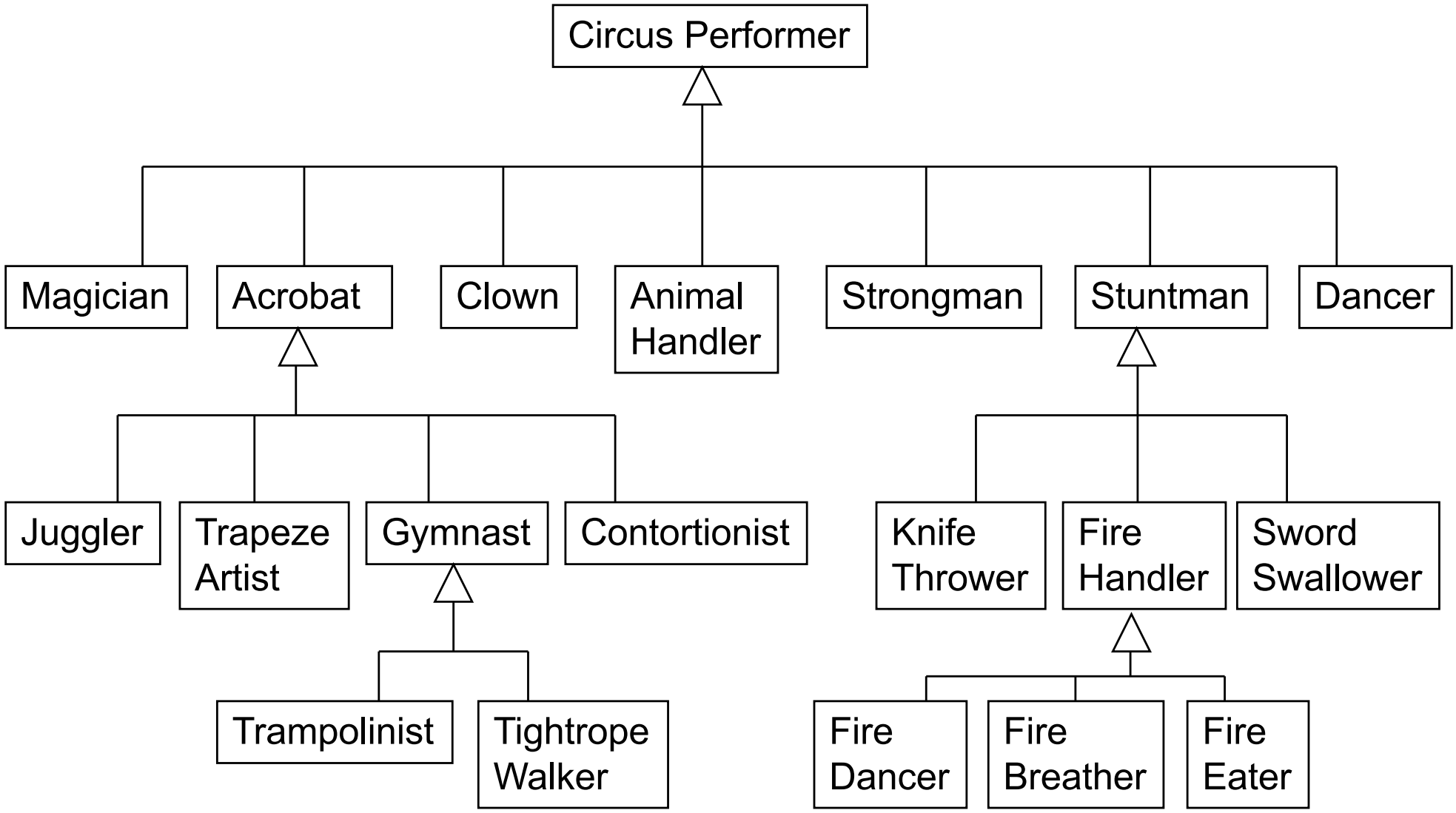
Inheritance

- Subclass “is a type of” superclass
- Subclass “is a” superclass
- The subclass “inherits” the properties of the superclass (base class)

Example

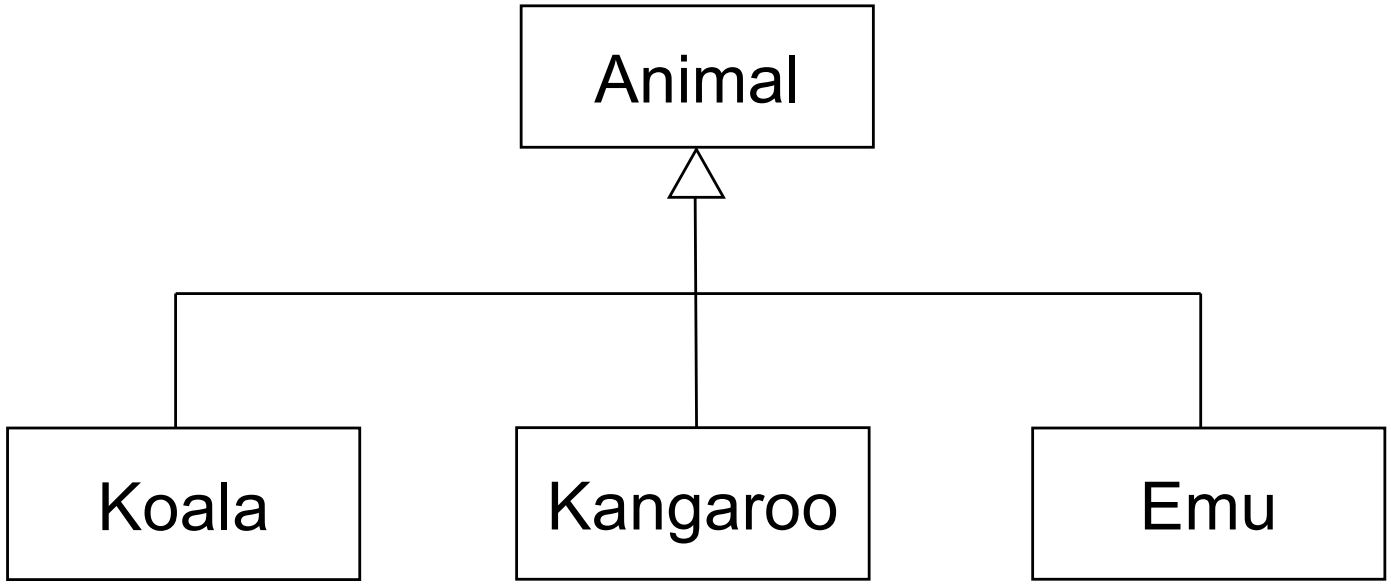

Person
-name
-address
-phoneNumber
-dateOfBirth
+changeAddress()
+getName()
+getAddress()
+getPhoneNumber()
+getDateOfBirth()
+calculateAge()
Student
-studentNumber
-course
-enrolledSubjects
+enrollInSubject()
+withdrawFromSubject()

A Class Hierarchy



Inheritance in Java

- As Java only allows single inheritance, any class has only one parent
- The class hierarchy forms a tree shape



Polymorphism

- Allows the same message to be sent to different types of objects with their own way of carrying out the requested operation
- E.g. send a “toast” message to a pop-up toaster or a roller toaster; each has an operation that “toasts” but the method of toasting is defined differently
- Polymorphism – “many forms”

3.6. Object-oriented vs procedural programming

Algorithms

- An algorithm is a set of unambiguous instructions defined to perform some task
- An algorithm can be expressed in a human language, or some form of code, diagram or programming language
- We often use pseudocode to describe an algorithm in an English-like manner (allowing transformation into different programming languages)

Procedural vs OO programming

- Procedural programming considers programs as a set of algorithms. These algorithms work on some data.
- OO programming views programs as a set of interacting objects which have their own states (attributes) and behaviour (methods)
- Algorithms are important in OO programming as the “body” of methods