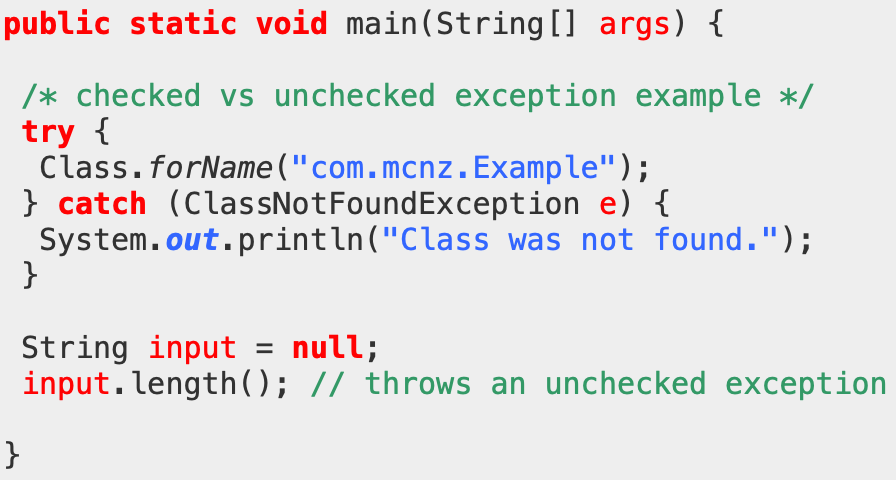
Name : Richard Reynard

JCU ID : 14170539

Date : 9 February 2023

Subject : CP2406 – Programming III

1. “If statements” evaluates integer, character, pointer or floating-point type or Boolean type. Whereas “switch statements” evaluates only character or an integer expression. Furthermore, multiple “if statement” are required for numerous decisions, whereas “switch statements” only need a single statement for numerous decisions. For the use case, “switch statement” is easier to be expressed for lengthy conditions because “if statement” will get more and more complex as the number of conditions grow (such as when using nested if). “Switch statements” can also be used to imitate a specific kind of if-else ladder involving a single variable whose value determine which part of the if-else ladder is executed. On the other hand, “if statements” can be used to control the execution flow in many more ways than a “switch statement” and “else” of “if-else” is similar to “default” case of a “switch statement”.
2. “Check exception” force the developer to handle the potential crash, used on critical resources such as files and database. Unchecked exceptions – the developer is not forced to handle the potential crash, typically associated with runtime issues (e.g. null pointer exceptions)



For example, in the code above, the Class.forName() method loads a class named “com.mcnz.Example”, but since the developer knows that the class my not exist, he is forced to handle the “ClassNotFoundException”, this is “Check exception” example.

Furthermore, in the input.length(), it will trigger NullPointerException and causes the program to fail when it runs. This “unchecked exception” will not be revealed until the program executes

1. Object is the parent class for all existing API classes in Java, as well as any developer created classes. The reason toString() is defined in Object is so that each child class has the opportunity to override their version of toString() for their specific situation and member fields.
2. Naming conventions are very important as they help the developer with the readability of their code. This is useful during programming as well as longer term maintenance of code. Variable names, constants, and class names are nouns. Method names are verbs. Package names are typically taken from website URLs in reverse.
3. An automatic default construct is created by Java if the developer doesn’t provide any constructors. E.g. a version of the Person class using an automatic default constructor would look like this:  
   class Person {  
   String name; int age;  
   }

Whereas a version of Person with a default constructor would look like this:  
class Person {

String name; int age; Person() {

name = "unknown";

age = 0; }

}  
Not providing a constructor means that Java will assign the default values to the class member fields. E.g. for a string that is null, and for an int that is 0.

1. Primitive types – fixed size, limited range of values, single values  
   Various numerical whole number data types – int, short, long, byte – used for whole number scalar values, the choice depends on the range of values required  
   Floating point types – float, double – they support fraction numbers, much bigger range than most of the whole number types – useful for representing money.
2. The String API consists of several useful and related string manipulation functionality brought together in a single class type. It’s functionality allows the programming to process and construct strings in different ways – very useful since many types of program deal with textual information processing.  
   Capabilities – string splitting and joining, substring searching, capitalisation, lowercase/uppercase switching, etc.
3. The Scanner API is very useful for allowing a program to get user input, either via the keyboard of from a text file. It provides a collection of overloaded methods for checking if the input stream matches a certain type of textual information (e.g. hasInt(), etc. whole numbers, or hasDouble() floating point numbers, or plain text) as well as overloaded methods for reading a specific type of textual information (nextInt() and nextDouble(), etc)
4. Static allows the process() method to be called directly in main() – which is itself a static method. Static keyword allows a method to be called directly – without the need to create an object.

Primitive arrays are fixed-size arrays that hold a set of data of the same type – e.g. an array in integer values. In this code, the process() takes a primitive array as an argument – that means the formal parameter “data” and the local variable “data” in main() both refer to the same primitive array in memory. This is done because arrays might be massive in size and hence coping all those values would slow the execution. Process() is being used to reset all of the values in the data array to 0 values.

1. Abstract classes are used to represent a concept in Java. For example, “shape” or “vehicle” is a concept. By creating this abstract class, it signals to us that they are not “completely definable” objects. For instance, “shape” can be drawn as squares, triangles, rectangles and many more. But this does not tell us how to draw that particular “shape” and is also illogical to implement draw() method for the class “shape”  
   Non-abstract classes are normal, concrete classes which are well defined. For example, A Rectangle Is A Shape. A rectangle has a specific length and width and other properties that makes it possible to draw the rectangle.   
   Usually, a class hierarchy contains abstract classes that are parents to non-abstract child classes. Abstract classes are partially defined. For example, aShapeCanMove(), means that the shape might have position and movable, but not drawable.
2. An interface is like completely abstract class – it’s a group of methods that represent some behaviour that a class could implement. Java doesn’t offer multiple inheritance – instead it allows a class to implement any number of interfaces to take on any number of behaviours.
3. **Private** will only allow access within the class, and will not allow access from outside of the class. Private class is frequently used to create a fully encapsulated class, in which it means that we are only interested in using that class in that particular main class. One thing to take note is that we cannot have private class it its own class file as it will show error. However, we can implement private class for the case of nested classes

**Protected** allows access from within the package and outside the package through child class. Therefore, if a child class is not created beforehand, it will be able to be accessed from outside the package.

For example, in the code below:

class Person {

protected String fname = "John";

protected String lname = "Doe";

}

class Student extends Person {

private int graduationYear = 2018;

public static void main(String[] args) {

Student myObj = new Student();

System.out.println("Name: " + myObj.fname + " " + myObj.lname);

System.out.println("Graduation Year: " + myObj.graduationYear);

}

}

The class Student extends Person, in which it inherits the attributes from class Person, where is has first name and last name of the student. Since the first and last name are set to protected, hence within the class Student, it will be able to access it. It’s usability is limited to cases where the developer is only interested for the class within the main() function to access the attributes or methods.

**Public** allows access from everywhere – within the class, outside of the class, within package or even outside of the package. This is the most flexible out of all access specifiers, as it can take attributes and methods from anywhere in the package. The example can be seen below:

// Main.java

Public class Main {

int x = 5  
}

// Second.java  
class Second {  
 public static void main(String[] args) {  
 Main myObj = new Main();  
 System.out.println(myObj.x);

}  
}

On the example above, we can see that the class Second (from Second.java) uses the value of “x” that is derived freely from the Main class (from Main.java), and it does not matter if they belong to the same package or not as they are set to public. This is particularly useful if there are many methods that have been created and the developer would like to reuse the methods on another project, and hence be able to simply call them, which significantly helps in reducing the coding time required as they will not have to do the same work twice.

1. Overloaded methods – contained within a single class definition, same method name but different method signatures.

Overridden methods – across a based class and a derived class, have exactly the same name and signature

1. “This” keyword is used to access methods of the current class and is reserved keyword in java, hence cannot be used as identifier. “This” is used to refer to the current class’s instance and static members, invoke current class constructor, be passed in as argument in method or constructor call and return the current class instance.  
     
   For example:

public class FixDebugBox

{

private int width;

private int length;

private int height;

public FixDebugBox()

{

length = 1;

width = 1;

height = 1;

}

public FixDebugBox(int width, int length, int height)

{

this.width = width;

this.length = length;

this.height = height;

}

public void showData()

{

System.out.println("Width: " + width + " Length: " +

length + " Height: " + height);

}

public double getVolume()

{

double vol = length \* width \* height;

return vol;

}

}

In this case, the use of “this” keyword is referring to the current class instance (“FixDebugBox”), and the instance variables are “width”, “length” and “height” and each of them is stored the value “1”. By constructing additional main(), the program will then be able to run, and shows the width, length and height (from showData()) and the volume of the box (from getVolume()).

“Super” keyword is used to access methods of the parent class, and similar to “this” keyword, “super” is also a reserved keyword in java and hence not able to be used as identifier. “Super” is used to refer super-class’s instance as well as static members, invoke super-class’s method or constructor. This is particularly useful as it eliminates the confusion between the super classes and subclasses that have methods with the same name. Furthermore, “Super” is useful to refer to immediate parent class instance variable, method and class constructor.

// Book.java

public abstract class Book

{ String title = new String();

double price;

public Book(String t)

{ title = t;}

public String getTitle()

{ return title; }

public double getPrice()

{ return price;}

public abstract void setPrice();}

// Fiction.java

public class Fiction extends Book { public Fiction(String title)

{ super(title);

setPrice();}

public void setPrice()

{ super.price = 24.99;}}

As seen from example above, Book.java is the parent class, whereas Fiction.java is the child class. In the child class, it uses “super.price” on the setPrice() such that it will be able to pass on the “price” of the book (eg. 24.99) to the main() when the code is run because it is accessing the method setPrice() from the parent class Book.java

1. The keyword “extends” indicates that a class is inherited from another class, which means that if a class is inherited from another class, it would also be possible to inherit the attributes and methods from that class. For this example, we can see that class B extends A, which means that class B will inherit the methods and attributes from class A.

Firstly, the main() will execute class A since class A comes first before anything else (as seen in **new A()**), in which it will then print the letter “A”.

Next, it will go to class B (**new B()**). Since class B “extends” class A, it means that it will then execute class A again, followed by class B, to which it will then print “A” and then “B”.

Then, it will go to class C (**new C()**), and since class C extends B and class B extends A, it means that it will first execute A, B and then C. Therefore, it will first print “A”, followed by “B” and lastly “C”

Hence, by combining the results – “A”, “AB” and “ABC”, the final output will be **“AABABC”**