CET325

Advanced Mobile Development

Session 1B

Java – In summary

- Java is a platform-independent language
  - The output of Java compiler is NOT executable!
  - It's bytecode.
  - This bytecode is then executed by a Java Virtual Machine (JVM), which interprets the byte code.

Hello World!

```
public class HelloWorld {
        public static void main(String[] args) {
            System.out.println("Hello World");
        }
}
```

- Class declaration
- Main method declaration
- Variable

Defining Classes

```
public class Book {
    private int id = 0;
    // keyword final replaces C# const
    public final int NULL_ID = -1;
    private static int totalNoBooks = 0;

    public Book(int mId){
        this.id = mId;
        totalNoBooks ++;
    }
}
```

Class accessibility

Fields with various accessibility and mutability

Constructor

Static fields

Class Properties (1)

- In C# you can declare auto generated accessor and mutator methods in your class property declaration:

```
private string FirstName{ get; set; }
private string LastName{ get; set; }

private string FullName{
        get{ return firstName + " "
                + lastName;}
}
```

Class Properties (2)

- In java you must declare accessor and mutator methods yourself:

```
// Note String type as object
private String firstName;

public void setFirstName( String name ) {
        this.firstName = name;
}

public String getFirstName() {
        return firstName;
}
```

What is this code doing?

```
public class SimpleExample {

private int number;

public SimpleExample() { }

public SimpleExample(int val) {
        number = val;
 }

public void setValue(int val) {
        number = val;
}

public int getNumber() {
        return number;
}
```

Simple Example

```
public static void main(String[] args) {

        for(int i=0;i<10;i++) {
          // instantiate new object
          SimpleExample example = new SimpleExample();

          if(i <= 5 ) {
                example.setValue(i);
          } else {
                example.setValue(i*10);
          }
          System.out.println("SimpleExample #"+i+
                "'s value is "+example.getNumber());
          }
        }
}
```

Simple Example - Output

```
SimpleExample #0's value is 0
SimpleExample #1's value is 1
SimpleExample #2's value is 2
SimpleExample #3's value is 3
SimpleExample #4's value is 4
SimpleExample #5's value is 5
SimpleExample #6's value is 60
SimpleExample #7's value is 70
SimpleExample #8's value is 80
SimpleExample #9's value is 90
```

Data Types: Primitives and Objects

- Java is a strongly typed language
  - You cannot let compiler determine type.
- Primitives:
  - boolean, byte, short, int, long, float, double, char
- Every other data type is an object
  - Including String.
  - Will have associated properties and methods

```
String myString = “Welcome”
int length = myString.length();
```

Operators

postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Basic Constructs - Selection

```
if(i/2 <= 2) {
        example.setValue(i);
} else {
        example.setValue(i*10);
}
```

Basic Constructs - Selection

```
int somenumber = 0;
// some logic changes somenumber's value making it
either 0, 1, or 2
switch(somenumber) {
        case 0:
                doSomething();
                break;
        case 1:
                doSomethingOne();
                break;
        case 2:
                doSomethingTwo();
                break;
}
```

Basic Constructs - Iteration - While

```
//-- the while loop
int i = 0;
// What will be the last print statement of this loop?
while(i < 10) {
        System.out.println(String.valueOf(i));
        i++;
}
```

Basic Constructs - Iteration – Do While

```
//-- the do while loop
int k = 0;
// this statement will be executed at least once, no
matter what the condition
do {
        System.out.println(String.valueOf(k));
        k++;
} while(k < 10);
```

Basic Constructs - Iteration - For

```
//-- the for loop
// the loop initializes j to 0, then increments it by +1
// (the j++) until j is equal to or greater than 10
for(int j=0;j<10;j++) {
        System.out.println(String.valueOf(j));
}
```

Basic Constructs - Branching

- Branching statements: break, continue, return

```
// forloop1
for(int i=0;i<10;i++) {
        // if i is even then continue to the next iteration of forloop1
        if(i%2 == 0) continue;
        else {
                // forloop2
                for(int j=0;j<5;j++) {
                        // if j%i has no remainder then jump out of forloop2
                        //and back to  forloop1
                        if(j%i != 0) break;
                        else return i;
                        // else return the integer value I and then stops the
                        complete flow
                }
        }
}
```

Arrays

```
double[] someArray; // declaring
someArray = new double[4]; // assigning size of 4

int[] integerArray = new int[5]; // declaring and assigning size of 5

integerArray[0] = 32; // assigning the first element
integerArray[1] = 12;
integerArray[2] = 333;
integerArray[3] = 3343;
integerArray[4] = 1;

// declaring and assigning 3 elements directly
String[] anotherArray = {"Some String","a","strings"}
```

Basic Constructs - Iteration – For Each

```
//-- for each loop
String[] arr = {"The","Quick","Brown","Fox"};
// the loop iterates through the entire array
for(String a: arr) {
        System.out.println(a);
}
```

Task – 5 minutes

- You are developing software for an animal shelter, and they want to be able to represent the animals they are caring for. Create an appropriate class to represent the following:

- Name
- Type
- Age
- Whether or not the animal has any known illnesses.
- The total animal count.

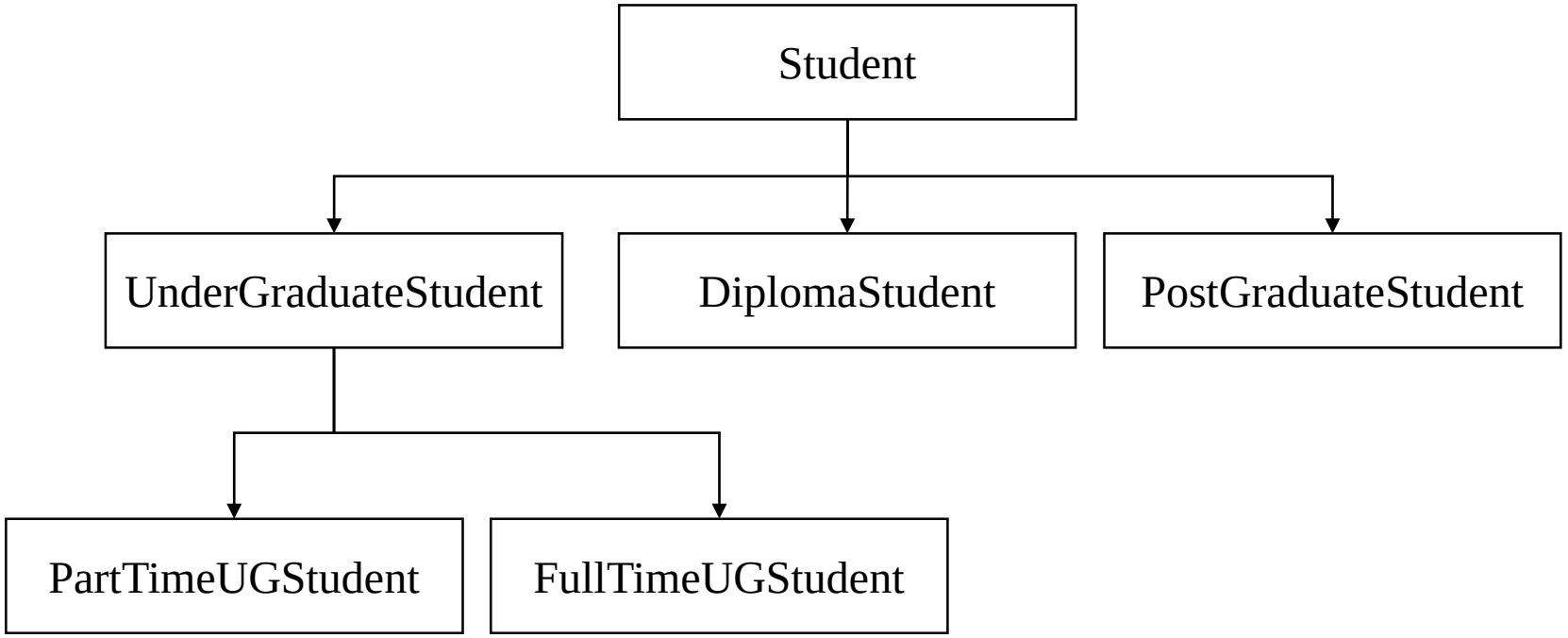
- What might a class constructor look like?

```java
public class Animal {

    private String name;
    private String type;
    private int age;
    private boolean hasKnownCondition;

    private static int totalAnimalCount;

    public Animal(){
        totalAnimalCount++;
    }

    public Animal(String pName, String pType,
       int pAge, boolean pHasCondition){
        name = pName;
        type = pType;
        age = pAge;
        hasKnownCondition = pHasCondition;
        totalAnimalCount++;
    }

    //Then declare get and set methods
}
```

Inheritance

- Software Reuse
  - A new class can be written without having to write all the data members and methods which are common to a parent (or super) class
  - The existing class is known as the *superclass or base or parent*
  - The new class is known as the *subclass or derived or child*

Inheritance



Inheritance in Java

```
class  Child extends Parent
{
  // provide only methods and fields
  // which differ from Parent
 }
```

Special considerations

- Constructors are not inherited
- We can over-ride methods belonging to the superclass.

Inheritance in Java

```
class  Child extends Parent
{
  // provide only methods and fields
  // which differ from Parent

  public Child () {
         super();
         }
 }
```

Inheritance in Android Studio

```
public class MainActivity extends ActionBarActivity {
    // In java the child can only extend ONE parent
    // i.e. multiple inheritance is not allowed
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // Constructor for the subclass should make use of
        // constructor for superclass by explicit call (super)
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the
        // action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
```

Abstract Classes and Methods

An abstract class is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

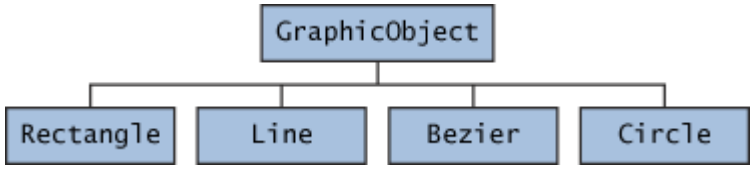
An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

```
abstract void moveTo(double deltaX, double deltaY);
```

If a class includes abstract methods, then the class itself must be declared abstract, as in:

```
public abstract class GraphicObject {
   // declare fields
   // declare nonabstract methods
   abstract void draw();
}
```

Abstract Classes and Methods




```
abstract class GraphicObject {
    int x, y;
    ...
    void moveTo(int newX, int newY) {
        ...
    }
    abstract void draw();
    abstract void resize();
}

class Circle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
```

Interfaces

- an *interface* is a reference type, similar to a class, that can contain *only* constants, method signatures, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.
- Interfaces cannot be instantiated—they can only be *implemented* by classes or *extended* by other interfaces.

Interfaces

```
interface Bicycle {
    public void changeGear(int newValue);
    public void speedUp(int increment);
    public void applyBrakes(int decrement);
}

class RoadBike implements Bicycle {

    int speed = 0;
    int gear = 1;

    public void changeGear(int newValue) {
         gear = newValue;
    }

    public void speedUp(int increment) {
         speed = speed + increment;
    }

    public void applyBrakes(int decrement) {
         speed = speed - decrement;
    }
```

Interfaces and Abstract Classes

- Abstract classes are similar to interfaces. You cannot instantiate them, and they may contain a mix of methods declared with or without an implementation.
- However, with abstract classes, you can declare fields that are not static and final, and define public, protected, and private concrete methods.
- You can extend only one class, whether or not it is abstract, whereas you can implement any number of interfaces.

Enum Types

- An *enum type* is a special data type that enables for a variable to be a set of predefined constants.

```
public enum Day {
 SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}


public class EnumTest {
    Day day;

    public EnumTest(Day day) {
        this.day = day;
    }
```

Enum Types

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27,   7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7);

    private final double mass;   // in kilograms
    private final double radius; // in meters
    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }
    private double mass() { return mass; }
    private double radius() { return radius; }
}

for (Planet p : Planet.values())
        System.out.printf("The mass of %s is %f%n",p,.mass());
    }
```

Error Handling

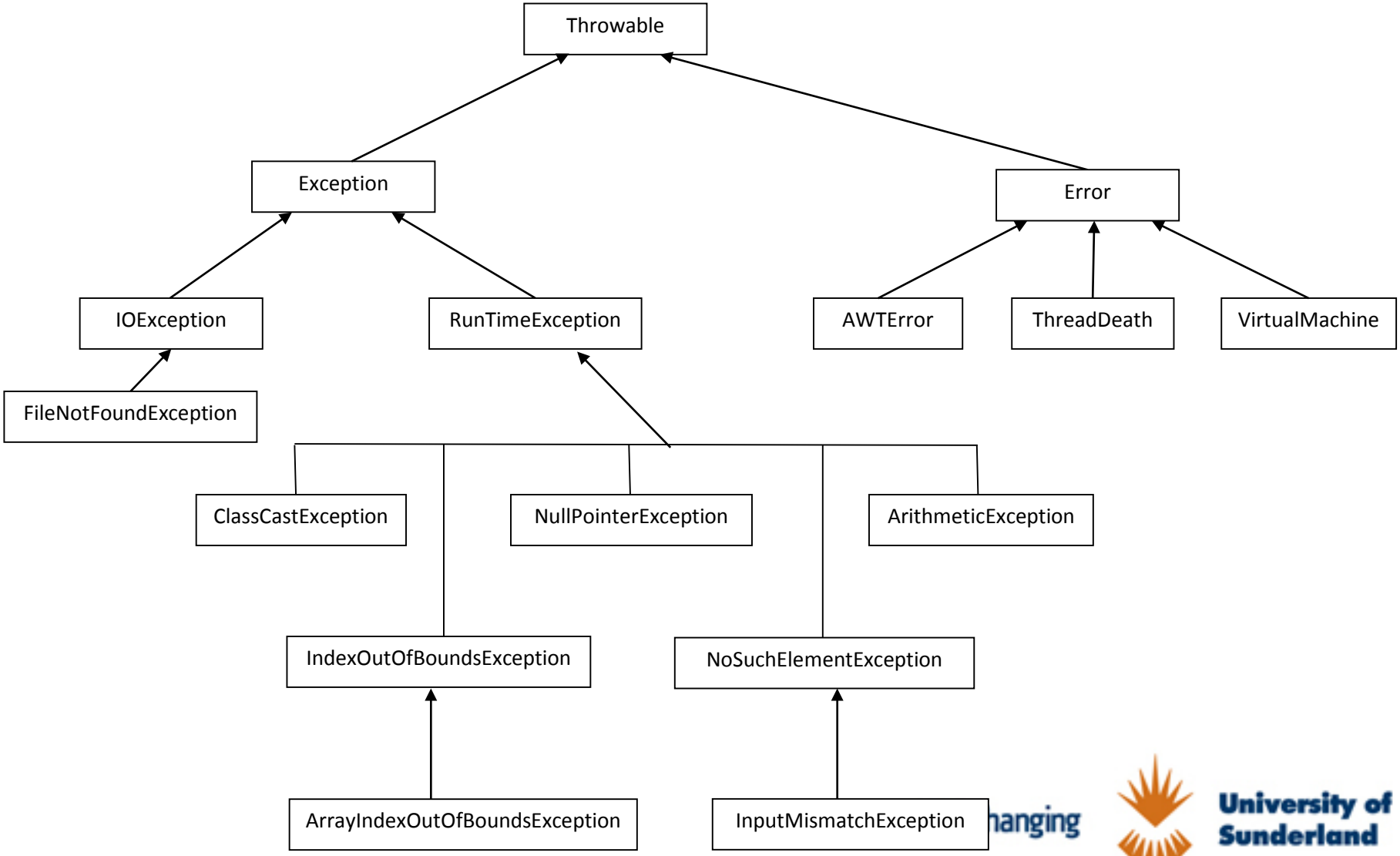
- Exceptions – errors that interrupt the normal flow of a program.

```
try {
   //statements-1
}
catch ( exception-class-name  variable-name) {
  // statements-2
 //  print error details
}
```

Try – Catch Examples

```
int sum = 123, count = 0;

try{
    int average = sum / count;
}
catch(ArithmeticException e){
    System.out.println(e.getMessage());
}
```

Error Handling



Try - Catch

- You can also make your own methods throw custom exceptions

```
public class SimpleExampleErrorHandling {

    private int number;

    public SimpleExampleErrorHandling() { }

    //------- ERROR HANDLING
    public void setValue(int val) throws Exception {

        if(val < 0) throw new Exception(
                "setValue Exception- Value is Negative!");
        number = val;
    }

    public int getNumber() {
        return number;
    }
}
```

Try – Catch Examples

```
FileReader reader = null;

try{
    //create, open and read from file
}
catch(IOException e){
    System.out.println(e.getMessage());
}
finally{
    // don't forget to close the file.
    if(reader != null){
        try {
            reader.close();
        } catch (IOException e) {
            //do something with the exception
        }
    }
    System.out.println("--- File End ---");
}
```

Recap

- Java is an O-O language, just like C#
  - The principles you have already learnt still apply.

- It's reasonable enough for purposes of this module to assume the only real difference to consider is Syntax:
  - You have examples of all the main programming constructs in this lecture.
  - Use it as your reference in the labs!