EE2-12 Software Engineering 2: Object-Oriented Programming

Week 9 - Java

Sahbi Ben Ismail

Imperial College London, Department of Electrical and Electronic Engineering

Imperial College London

Autumn Term 2018-19

Module Syllabus

- Week 1 Classes and Objects I: Introduction
- Week 2 Classes and Objects II: Constructors, and Operator Overloading
- Week 3 More on Classes, Objects, and Operator Overloading
- Week 4 Objects and Dynamic Memory
- Week 5 Classes Relationships: Association, Aggregation/Composition and Generalisation (Inheritance)
- Week 6 Polymorphism and Virtual Functions
- Week 7 Generic Programming: Templates, and the Standard Template Library (STL)
- Week 8 Exceptions Handling
- Week 9 **C++ to Java**
- Week 10 Revision

Week 9 Intended Learning Outcomes (ILOs)

By the end of this week you should be better able to:

Lecture

1. Apply the key OOP concepts in the Java programming language (**abstraction**, **encapsulation**, **inheritance**, **polymorphism**) ('Big Four')
2. Understand the main **common features** and **differences** between C++ and Java
3. Write classes in Java using **genericity** and **exceptions**

(No Lab this week)

Problematic: C++ to Java

- Week 1 Classes and Objects I: Introduction
- Week 2 Classes and Objects II: Constructors, and ~~Operator Overloading~~
- Week 3 More on Classes, Objects, and ~~Operator Overloading~~
- Week 4 Objects and ~~Dynamic Memory~~
- Week 5 Classes Relationships: Association, Aggregation/Composition and Generalisation (Inheritance)
- Week 6 Polymorphism and ~~Virtual Functions~~ abstract classes/interfaces
- Week 7 Generic Programming: Templates, and ~~the Standard Template Library (STL)~~ Collections
- Week 8 Exceptions Handling

Outline

1. Introduction
   - Week 8 Exceptions - Summary
2. C++ to Java
   - Objects and Classes
   - Composition
   - Inheritance and Polymorphism
   - Genericity, Collections and Exceptions
3. Conclusion

Outline

1. Introduction
   - Week 8 Exceptions - Summary
2. C++ to Java
3. Conclusion

Summative assessments - Recap

1-h Lab Assessment: Wed 12/12/2018 9:30-10:30

- closed book: only online resource allowed is the module Blackboard page (lectures, labs, Makefiles)
- you can have access to your own Labs code
- content: weeks 1 to 8 (C++ only, no Java)
- Output: BB submission of a **studentId.zip** archive
  - README file (how to build/run your application)
  - Makefile
  - source code (*.hpp and *.cpp)
- Advice: save, compile & debug. do not 'waste' time in search/copy/paste

Final year written 2-h exam: Mon 03/06/2019 10:00-12:00

- Revision Lecture in May 2019 (**Thursday 09th 10:00-12:00** TBC)
- content: ALL weeks 1 to 9 (C++ & Java)

Week 8 - Exceptions Handling

1. How to throw exceptions
   How to catch exceptions
2. How to write user-defined exceptions
3. How to catch exceptions in the right order
4. RAII idiom (*Resource Acquisition is Initialization*)

How to catch exceptions

```
//try-catch block
try{
  //functions calls
  //...
  }
  catch(const exception_type& e){
    cout << "exception caught" << e (or e.function()) << ...
    //processing
    //...
  }
```

How to throw exceptions - C-string I

```
double my_sqrt(double n);

int main() {
  double n;
  std::cout << "Enter a positive number n:" << std::endl;
  std::cin >> n;

  try {
    std::cout << my_sqrt(n) << std::endl;
  }
  catch (const char*& e) {
    std::cout << "exception thrown and caught (char*): " << e << std::endl;
  }

  std::cout << "bye." << std::endl;

  return 0;
}


double my_sqrt(double n) {
  if(n<0) {
    throw "negative number";
  }
  return sqrt(n);
}
```

How to throw exceptions - C-string II

```
/* Output:
  $ ./prog
  Enter a positive number n:
  -1
  exception thrown and caught (char*): negative number
  bye.
*/
```

How to throw exceptions - std::string I

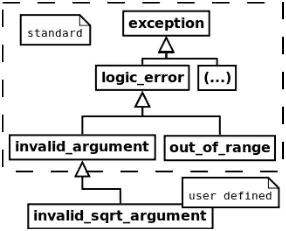
```cpp
double my_sqrt(double n);

int main() {
  double n;
  std::cout << "Enter a positive number n:" << std::endl;
  std::cin >> n;

  try {
    std::cout << my_sqrt(n) << std::endl;
  }
  catch (const std::string& e) {
    std::cout << "exception thrown and caught: (std::string): " << e << std::endl;
  }

  std::cout << "bye." << std::endl;

  return 0;
}


double my_sqrt(double n) {
  if(n<0) {
    throw std::string("negative number");
  }
  return sqrt(n);
}
```

How to throw exceptions - std::string II

```
/* Output:
  $ ./prog
  Enter a positive number n:
  -1
  exception thrown and caught: (std::string): negative number
  bye.
*/
```

How to write user-defined exceptions

```
class invalid_sqrt_argument : public invalid_argument {
    public:
        invalid_sqrt_argument(const string& what) :
            invalid_argument(what) {}
};
```



How to catch exceptions in the right order

- starting from the most specific (only the first matched exception is caught)

```
catch(const invalid_sqrt_argument& e){
     cout << "unsuccessful sqrt computation: " << e.what() << e
 }
 catch(const out_of_range& e){
     cout << "index not suitable: " << e.what() << endl;
 }
 catch(const logic_error& e){
     cout << "some other logic error: " << e.what() << endl;
 }
 catch(const exception& e){
     cout << "some other exception" << endl;
 }
```

RAII idiom (*Resource Acquisition is Initialization*)

How to ensure all resources (files, databases, sockets etc ...) are acquired & released properly

```cpp
class class_name {
  public:
    class_name(); //Constructor
    //acquires the resource and throws exceptions on errors in handling it

    ~class_name(); //Destructor
    //releases the resource, removing the need for an explicit clean-up on the user's end

    //...

};
```

Outline

1. Introduction
2. C++ to Java
   - Objects and Classes
   - Composition
   - Inheritance and Polymorphism
   - Genericity, Collections and Exceptions
3. Conclusion

History

C++

- 1970's (**C++98**, ..., **C++11**, C++14, C++17)
- hybrid: C + OO
- **performance**

Java

- 1990's (Java 1 in 1994, Java 5 in 2004, **Java 8** (LTS) in 2014)
- pure OO
- **portability**

Java - HelloWorld!

```
public class HelloWorld {

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

```
/* Output:
  $ ls
  HelloWorld.java
  $ javac HelloWorld.java
  $ ls
  HelloWorld.class  HelloWorld.java
  $ ./HelloWorld.class
  -bash: ./HelloWorld.class: cannot execute binary file:
      Exec format error
  $ java HelloWorld
  Hello world!
*/
```

Java

- Everything is in a class.
- (Not everything is an object e.g. `int`, `double` are still just primitive types.)
- Usually only one class per file (and file name equal to class name) (it's more complicated than this and there is also a concept of class visibility and "package", but follow the guideline).
- Static functions (called on classes rather than on objects) (exist also in C++).
- `main`: `void` return value; parameters are needed (even if not used).
- `javac` compiler compiles to bytecode (not a full executable).
- `java` Java Virtual Machine (JVM) executes the program.
- (Member functions are called "methods", member data "attributes".)

Point.java I

```java
import java.lang.Math; //for sqrt() and pow()
//API: https://docs.oracle.com/javase/10/docs/api/java/lang/Math.html

public class Point {
  //Fields (or UML/Attributes) (C++/Variables)
  private double x;
  private double y;

  //Constructor
  public Point(double x_in, double y_in) {
    x = x_in;
    y = y_in;
  }

  //Methods (or UML/Operations) (C++/Functions)
  public double getX() {
    return x;
  }

  public void setX(double x_in)  {
    x = x_in;
```

Point.java II

```java
  }

  public double getY() {
    return y;
  }

  public void setY(double y_in)  {
    y = y_in;
  }

  public void translate(double deltaX, double deltaY) {
    x += deltaX;
    y += deltaY;
  }

  public double distanceTo(Point other) {
    double deltaX = x - other.x;
    double deltaY = y - other.y;
    double distance = Math.sqrt(Math.pow(deltaX, 2) + Math.
        pow(deltaY, 2));

    return(distance);
  }
```

Point.java III

```java

  public double distanceToOrigin() {
    Point origin = new Point(0, 0);
    return distanceTo(origin);
  }

  public boolean equals(Point p){
    return (x == p.x) && (y == p.y);
  }

  public String toString() {
    return("(" + x + ", " + y + ")");
  }
}
```

Point.java - Notes

- Overloading (no default parameters – there are other kinds of defaults).
- No initialization list.
- Everything passed “by reference” (more on this later).
- Visibility identifiers prefixed to each field.
- No operator overloading but conventional method names.

PointTest.java I

```java
//no need to import class Point
//(files Point.java and PointTest.java are in the same folder)

public class PointTest {
  public static void main(String[] args) {
    Point p1 = new Point(1, 2);
    System.out.println("Point p1: " + p1.toString());
    System.out.println("distanceTo(p1): " + p1.distanceTo(p1));

    p1.translate(2, 2);

    System.out.println("after translate(2, 2): " + p1.toString());
    System.out.println("distanceToOrigin(): " + p1.distanceToOrigin());

```

PointTest.java II

```
    Point p2 = new Point(4, 4);

    System.out.println("p1.distanceTo(p2): " + p1.distanceTo
        (p2));
    System.out.println("p2.distanceTo(p1): " + p2.distanceTo
        (p1));
  }
}
```

```
/* Output:
  $ javac PointTest.java
  $ java PointTest
  Point p1: (1.0, 2.0)
  distanceTo(p1): 0.0
  after translate(2, 2): (3.0, 4.0)
  distanceToOrigin(): 5.0
  p1.distanceTo(p2): 1.0
  p2.distanceTo(p1): 1.0
*/
```

PointTest.java - Notes I

- `Point p2` declaration: `p2` is a sort of pointer or reference to `Point` (a "handle").
- Objects are always constructed dynamically.
- The JVM keeps a reference count of memory areas.
- Periodically the garbage collector deallocates memory areas which are not referenced anymore.
- No more concerns about memory leaks.
- No "raw pointers" (but handles can still be set to `null`. . .).
- Overhead penalizes performance.

PointTest.java - Notes II

- + operator to concatenate Strings.
- System.out.println() overloaded for primitive types, Strings and Objects.
- All classes in Java inherit from Object (which has, for instance, methods equals and toString).
- “Always override hashCode if you override equals.”
- Compiler automatically finds and compiles dependencies (if in specific locations – the same directory is one of them).

More on Point.java

- Java API specification
  https://docs.oracle.com/javase/8/docs/api/
- this: attributes or methods (including constructors): this.x, this.distanceTo(), this()
- toString(): method overriding (inherited from base class java.lang.Object)
- javadoc comments
- class attributes/methods (static)
- equals() and compareTo()

Point.java - this

```java
  public Point() {
    this(0.0, 0.0);
  }

  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }

  //Methods (or UML/Operations) (C++/Functions)
  public double getX() {
    return x; //or return this.x;
  }
```

Point.java - toString() I

```java
//no need to import class Point
//(files Point.java and PointTest.java are in the same folder)

public class PointTest {

  public static void main(String[] args) {
    Point p1 = new Point(1, 2);
    System.out.println(p1.toString());
    System.out.println(p1); //toString() inherited from class java.lang.Object

    //System.out.println("Point p1: " + p1.toString());
    System.out.println("Point p1: " + p1);
    System.out.println("distanceTo(p1): " + p1.distanceTo(p1));

    p1.translate(2, 2);
```

Point.java - toString() II

```

    //System.out.println("after translate(2, 2): " + p1.toString());
    System.out.println("after translate(2, 2): " + p1);
    System.out.println("distanceToOrigin(): " + p1.distanceToOrigin());

    Point p2 = new Point(4, 4);

    System.out.println("p1.distanceTo(p2): " + p1.distanceTo(p2));
    System.out.println("p2.distanceTo(p1): " + p2.distanceTo(p1));
  }
}
```

Point.java - toString() III

```
/* Output:
  $ java PointTest
  (1.0, 2.0)
  (1.0, 2.0)
  Point p1: (1.0, 2.0)
  distanceTo(p1): 0.0
  after translate(2, 2): (3.0, 4.0)
  distanceToOrigin(): 5.0
  p1.distanceTo(p2): 1.0
  p2.distanceTo(p1): 1.0
*/
```

Point.java - Javadoc comments I

```java
import java.lang.Math; //for sqrt() and pow()
//API: https://docs.oracle.com/javase/10/docs/api/java/lang/Math.html

/**
* The class Point models a 2D point in a cartesian plan having two coordinates x and y.
* A point can be translated, and can calculate its distance to another point.
* @author Sahbi Ben Ismail
* @version 1.0
* @since Autumn Term 2018-19 (EE2-12)
*/
public class Point {
  //Fields (or UML/Attributes) (C++/Variables)
  /**
  * First coordinate (abscissa).
  */
  private double x;

  /**
  * Second coordinate (ordinate).
```

Point.java - Javadoc comments II

```java
  */
  private double y;

  //Constructors
  /**
  * Constructs a point with coordinates (0.0, 0.0).
  */
  public Point() {
    this(0.0, 0.0);
  }

  /**
  * Constructs a point with two given coordinates.
  * @param x first coordinate (abscissa)
  * @param y second coordinate (ordinate)
  */
  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }

  //Methods (or UML/Operations) (C++/Functions)
  /**
```

Point.java - Javadoc comments III

```
  * Returns the first coordinate this.x (abscissa).
  * @return x
  */
  public double getX() {
    return x; //or return this.x;
  }

  /**
  * Modifies the value of the first coordinate this.x (abscissa).
  * @param x the new value of this.x
  */
  public void setX(double x)  {
    this.x = x;
  }

  /**
  * Returns the second coordinate this.y (ordinate).
  * @return y
  */
  public double getY() {
    return y; //or return this.y;
  }
```

Point.java - Javadoc comments IV

```java

  /**
  * Modifies the value of the second coordinate this.y (ordinate).
  * @param y the new value of this.y
  */
  public void setY(double y)  {
    this.y = y;
  }


  /**
  * Translates the point.
  * @param deltaX translation on the x axis
  * @param deltaY translation on the y axis
  */
  public void translate(double deltaX, double deltaY) {
    x += deltaX;
    y += deltaY;
  }

  /**
```

Point.java - Javadoc comments V

```java
    * Returns the distance between the current point and another one.
    * @param other point to which the distance is calculated
    * @return distance the calculated distance
    */
    public double distanceTo(Point other) {
      double deltaX = this.x - other.x;
      double deltaY = this.y - other.y;
      double distance = Math.sqrt(Math.pow(deltaX, 2) + Math.pow(deltaY, 2));

      return(distance);
    }

    /**
    * Returns the distance between the current point and the origin point (0.0, 0.0).
    * @return the calculated distance
    */
    public double distanceToOrigin() {
      Point origin = new Point(0, 0);
      return distanceTo(origin); //or this.distanceTo(origin)
    }
```

Point.java - Javadoc comments VI

```java

  /**
  * Returns a description of the point.
  */
  public String toString() {
    return("(" + x + ", " + y + ")");
  }
}
```

```
/* Ouput:
  $ mkdir doc
  $ javadoc Point.java -d doc/
  Loading source file Point.java...
  Constructing Javadoc information...
  Standard Doclet version 1.8.0_131
  Building tree for all the packages and classes...
  Generating doc\Point.html...
  Generating doc\package-frame.html...
  Generating doc\package-summary.html...
  Generating doc\package-tree.html...
  Generating doc\constant-values.html...
  Building index for all the packages and classes...
```

Point.java - Javadoc comments VII

```
  Generating doc\overview-tree.html...
  Generating doc\index-all.html...
  Generating doc\deprecated-list.html...
  Building index for all classes...
  Generating doc\allclasses-frame.html...
  Generating doc\allclasses-noframe.html...
  Generating doc\index.html...
  Generating doc\help-doc.html...

  $ javadoc -author Point.java -d doc/
  ...
*/
```

Point.java - class attributes/methods (static)

```java
  //Class fields
  public static int nb_points = 0; //total number of the Point instances (objects)

  //Constructors
  public Point() {
    this(0.0, 0.0);
  }

  public Point(double x, double y) {
    this.x = x;
    this.y = y;

    nb_points++;
  }
```

PointTest.java - class attributes/methods (static) I

```java
public class PointTest {

  public static void main(String[] args) {

    System.out.println("Point.nb_points = " + Point.nb_points);

    Point origin = new Point();

    System.out.println("Point.nb_points = " + Point.nb_points);

    Point p = new Point(1, 2);

    //a class field or method can be called from the class itself
    //or from any other instanciated object
    System.out.println("Point.nb_points = " + Point.nb_points);
    System.out.println("Point.nb_points = " + origin.nb_points);
    System.out.println("Point.nb_points = " + p.nb_points);
```

PointTest.java - class attributes/methods (static) II

```

    /* other examples of static fields/methods:
    class Math: https://docs.oracle.com/javase/10/docs/api/java/lang/Math.html
    Math.PI, Math.E
    Math.abs(), Math.min(), Math.max(), Math.sqrt(), Math.pow();

    class System: https://docs.oracle.com/javase/10/docs/api/java/lang/System.html#out
    System.out in System.out.println() (out is a static field of type PrintStream)

    class String: https://docs.oracle.com/javase/10/docs/api/java/lang/String.html
    */
  }
}
```

PointTest.java - class attributes/methods (static) III

```
/* Output:
  $ java PointTest
  Point.nb_points = 0
  Point.nb_points = 1
  Point.nb_points = 2
  Point.nb_points = 2
  Point.nb_points = 2
*/
```

Composition - Triangle.java I

```java
public class Triangle {
  private Point p1;
  private Point p2;
  private Point p3;

  public Triangle(Point p1, Point p2, Point p3) {
    this.p1 = p1;
    this.p2 = p2;
    this.p3 = p3;
  }

  public Triangle(double x1, double y1, double x2, double y2
      , double x3, double y3) {
    this.p1 = new Point(x1, y1);
    this.p3 = new Point(x2, y2);
    this.p3 = new Point(x3, y3);
  }

```

Composition - Triangle.java II

```java
  //getters
  //...

  //setters
  //...

  public void translate(double deltaX, double deltaY) {
    p1.translate(deltaX, deltaY);
    p2.translate(deltaX, deltaY);
    p3.translate(deltaX, deltaY);
  }

  public String toString() {
    return("Triangle[" + p1 + "->" + p2 + "->" + p3 + "]");
  }

}
```

Triangle (?)

- Notice that attributes are just handles (they are not objects being default constructed and then re-assigned).
- The assignment `this.p1 = p1` means something along the lines of “handle a now points to the same memory area pointed by handle p1”.

Triangle test - Main.java I

```java
public class Main {
  public static void main(String[] args) {
    Point p1 = new Point();
    Point p2 = new Point(0, 1);
    Point p3 = new Point(1, 0);

    Triangle t = new Triangle(p1, p2, p3);
    System.out.println(t);

    System.out.println("\np1.translate(-1, 0)");
    p1.translate(-1, 0);
    System.out.println(p1);
    System.out.println(t);

    System.out.println("\nt.translate(1, 1)");
    t.translate(1, 1);
    System.out.println(t);
    System.out.println(p1);
```

Triangle test - Main.java II

```
        System.out.println(p2);
        System.out.println(p3);
    }
}
```

```
/* Output:
  $ java Main
  Triangle[(0.0, 0.0)->(0.0, 1.0)->(1.0, 0.0)]

  p1.translate(-1, 0)
  (-1.0, 0.0)
  Triangle[(-1.0, 0.0)->(0.0, 1.0)->(1.0, 0.0)]

  t.translate(1, 1)
  Triangle[(0.0, 1.0)->(1.0, 2.0)->(2.0, 1.0)]
  (0.0, 1.0)
  (1.0, 2.0)
  (2.0, 1.0)
*/
```

Improved Triangle.java - Constructor

```java
  public Triangle(Point p1, Point p2, Point p3) {
    this.p1 = new Point(p1); //this.p1 = p1;
    this.p2 = new Point(p2); //this.p2 = p2;
    this.p3 = new Point(p3); //this.p3 = p3;
  }
```

Improved Point.java - Copy constructor

```java
public class Point {
  //Fields (or UML/Attributes) (C++/Variables)
  protected double x;
  protected double y;

  //Constructors
  public Point() {
    this(0.0, 0.0);
  }

  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public Point(Point p) { // copy constructor
    x = p.x;
    y = p.y;
  }
```

Improved Triangle/Point test - Main.java I

```java
public class Main {
  public static void main(String[] args) {
    Point p1 = new Point();
    Point p2 = new Point(0, 1);
    Point p3 = new Point(1, 0);

    Triangle t = new Triangle(p1, p2, p3);
    System.out.println(t);

    System.out.println("\np1.translate(-1, 0)");
    p1.translate(-1, 0);
    System.out.println(p1);
    System.out.println(t);

    System.out.println("\nt.translate(1, 1)");
    t.translate(1, 1);
    System.out.println(t);
    System.out.println(p1);
```

Improved Triangle/Point test - Main.java II

```
        System.out.println(p2);
        System.out.println(p3);
    }
}

/* Output:
  $ java Main
  Triangle[(0.0, 0.0)->(0.0, 1.0)->(1.0, 0.0)]

  p1.translate(-1, 0)
  (-1.0, 0.0)
  Triangle[(0.0, 0.0)->(0.0, 1.0)->(1.0, 0.0)]

  t.translate(1, 1)
  Triangle[(1.0, 1.0)->(1.0, 2.0)->(2.0, 1.0)]
  (-1.0, 0.0)
  (0.0, 1.0)
  (1.0, 0.0)
*/
```

Improved Triangle/Point test - Main.java III

↪ Triangle t is now ‘independant’ from Points p1, p2, and p3 used to construct it.

Improved Point.java - compareTo() I

```
public class Point implements Comparable<Point> {
```

```
  public int compareTo(Point p) {
        Point o = new Point();
        double dthis = distanceTo(o);
        double dp = p.distanceTo(o);

        if(dthis < dp) {
            return -1;
        }
        else if(dthis == dp) {
            return 0;
        }
        else {
            return 1;
        }
    }
```

Comparable Points

- `implements Comparable<T>` means that the class implements a certain `interface`.
- Interfaces are analogous to pure abstract classes (all methods abstract – equivalent of “pure virtual”).
- Classes can implement multiple interfaces.
- Implementing an interface means implementing the methods declared in it (“the class is (of a type) capable of performing these operations”).
- In this case that's method `int compareTo(T c)` (comparison to an object of the type specified in the `interface`.
- Call to `o1.compareTo(o2)` should return a negative integer if o1 is less than o2, positive if o1 is greater than o2 and 0 if they are equal.

Improved Point.java - Sorting - SortPoints.java I

```java
//import java.util.*; //imports all classes/interfaces in the package java.util

import java.util.Collections;
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;

public class PointsSort {
  public static void main(String[] args) {
    List<Point> points = new ArrayList<Point>();
    Scanner input = new Scanner(System.in);

    //get the points coordinates from the User
    for(int i = 0; i < 3; i++) {
      double x = input.nextDouble();
      double y = input.nextDouble();
```

Improved Point.java - Sorting - SortPoints.java II

```
      points.add(new Point(x,y));
    }

    for(int i = 0; i< 3; i++) {
      System.out.println(points.get(i));
    }

    //sort the list of points
    Collections.sort(points);

    //check the result
    System.out.println();
    Iterator itr = points.iterator(); //use of iterators
    while(itr.hasNext()) {
      System.out.println(itr.next());
    }
  }
}
```

Improved Point.java - Sorting - SortPoints.java III

```
/* Output:
  $ java PointsSort
  5
  4
  1
  2
  0
  1
  (5.0, 4.0)
  (1.0, 2.0)
  (0.0, 1.0)

  (0.0, 1.0)
  (1.0, 2.0)
  (5.0, 4.0)
*/
```

Inheritance - LabeledPoint.java I

```java
public class LabeledPoint extends Point { //inheritance
  private String label;

  public LabeledPoint() {
    super(); //call of the base class constructor
    label = "";
  }

  public LabeledPoint(double x, double y, String label) {
    super(x, y); //call of the base class constructor
    this.label = label;
  }

  public String getLabel() {
    return label;
  }

  public void setLabel(String label) {
```

Inheritance - LabeledPoint.java II

```
    this.label = label;
  }

  //method overriding
  public String toString() {
    return(label + super.toString());
  }
}
```

Inheritance - LabeledPointTest.java I

```java
public class LabeledPointTest {

  public static void main(String[] args) {
    //Point origin = new Point();
    Point p = new Point(1, 2);
    System.out.println(p);

    LabeledPoint lp = new LabeledPoint(2, 3, "A");
    System.out.println(lp.toString());
    System.out.println(lp);

    lp.setLabel("B");
    System.out.println(lp);
  }
}
```

Inheritance - LabeledPointTest.java II

```
/* Output:
  $ java LabeledPointTest
  (1.0, 2.0)
  A(2.0, 3.0)
  A(2.0, 3.0)
  B(2.0, 3.0)
*/
```

Polymorphism - Game I

```java
public abstract class Warrior {
  public abstract void attack(); //abstract method

  protected String name;

  public Warrior(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }
}
```

Polymorphism - Game II

```java
public class Ninja extends Warrior {
  public Ninja(String name) {
    super(name);
  }

  //implementation of the abstract method attack(), inherited from Warrior
  public void attack() {
    System.out.println("Ninja " + this.name + " attacks.");
  }
}
```

Polymorphism - Game III

```
public class Samurai extends Warrior {
  public Samurai(String name) {
    super(name);
  }

  //implementation of the abstract method attack(),
      inherited from Warrior
  public void attack() {
    System.out.println("Samurai " + this.name + " attacks.")
        ;
  }
}
```

Polymorphism - Game IV

```java
import java.util.Vector;
//https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html

//can also use java.util.ArrayList (roughly equivalent to Vector, except that it is unsynchronized.)
//https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

public class Main {
  public static void main(String[] args) {
    Vector<Warrior> v = new Vector<Warrior>();
    /*Ninja n1 = new Ninja("Ninja1");
    Ninja n2 = new Ninja("Ninja2");

    Samurai s1 = new Samurai*/

    v.add(new Ninja("Ninja1"));
    v.add(new Ninja("Ninja2"));

```

Polymorphism - Game V

```
    v.add(new Samurai("Samurai1"));
    v.add(new Samurai("Samurai2"));

    for(int i=0; i < v.size(); i++) {
      v.get(i).attack();
    }
  }
}

/* Output:
  $ java Main
  Ninja Ninja1 attacks.
  Ninja Ninja2 attacks.
  Samurai Samurai1 attacks.
  Samurai Samurai2 attacks.
*/
```

Polymorphism - Shape I

```
public abstract class Shape { //can also be defined as an interface
  public abstract double perimeter();
  public abstract double area();
}
```

```
public class Circle extends Shape {
  private Point centre;
  private double radius;

  public Circle() {
    this(new Point(), 1);
  }

  public Circle(Point centre, double radius) {
    this.centre = centre;
    this.radius = radius;
```

Polymorphism - Shape II

```java
  }

  //getters
  //...

  //setters
  //...

  public double perimeter() {
    return(2 * Math.PI * radius);
  }

  public double area() {
    return(Math.PI * Math.pow(radius, 2));
  }

  public String toString() {
    return("Circle[" + this.centre + ", " + this.radius + "]");
  }
```

Polymorphism - Shape III

```java
}
```

```java
public class Rectangle extends Shape {
  private Point topLeftCorner;
  private double width;
  private double height;

  public Rectangle() {
    this(new Point(), 1, 1);
  }

  public Rectangle(Point topLeftCorner, double width, double height) {
    this.topLeftCorner = topLeftCorner;
    this.width = width;
    this.height = height;
  }

  //getters
```

Polymorphism - Shape IV

```java
  //...

  //setters
  //...

  public double perimeter() {
    return(2*(width + height));
  }

  public double area() {
    return(width * height);
  }

  public String toString() {
    return("Rectangle[" + this.topLeftCorner + ", " + this.width + ", " + this.height + "]");
  }
}
```

Polymorphism - Shape V

```java
import java.util.Vector;

public class Main {
  public static void main(String[] args) {
    Vector<Shape> v = new Vector<Shape>();

    Shape s1 = new Circle();
    Shape s2 = new Rectangle();
    v.add(s1);
    v.add(s2);

    for(int i=0; i < v.size(); i++) {
      System.out.println("\n" + v.get(i));

      System.out.println("perimeter = " + v.get(i).perimeter());
      System.out.println("area = " + v.get(i).area());
    }
  }
}
```

Polymorphism - Shape VI

```
/* Output:
  $ java Main

  Circle[(0.0, 0.0), 1.0]
  perimeter = 6.283185307179586
  area = 3.141592653589793

  Rectangle[(0.0, 0.0), 1.0, 1.0]
  perimeter = 4.0
  area = 1.0
*/
```

Genericity I

```java
//Genericity (Java >= 1.5 (Java 5))

//generic class
public class Box<T> {
   private T t;

   public void add(T t) {
      this.t = t;
   }

   public T get() {
      return t;
   }

   public static void main(String[] args) {
      Box<Integer> integerBox = new Box<Integer>();
      Box<String> stringBox = new Box<String>();

```

Genericity II

```
      integerBox.add(new Integer(10));
      stringBox.add(new String("Hello World"));

      System.out.printf("Integer Value :%d\n\n", integerBox.get());
      System.out.printf("String Value :%s\n", stringBox.get());
   }
}

/* Output:
  $ java Box
  Integer Value :10

  String Value :Hello World
*/
```

Collections

https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html
https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html

Interface `java.util.Collection`

- `Interface Collection<E>`
- ArrayList<E>
- LinkedList<E>
- Vector<E>

Class `java.util.Collections`

- sort
- binarySearch
- swap
- min, max etc...

Exceptions - NullPointerTest.java I

```java
public class NullPointerTest {
  public static void main(String[] args) {
    try {
      String s = null;
      s.length();
      /*Point p = null;
      p.getX();*/
    }
    catch (Exception e) {
      System.out.println(e);
      e.printStackTrace();
    }

    System.out.println("\nEnd.");
  }
}
```

Exceptions - NullPointerTest.java II

```
/* Output:
  $ java NullPointerTest
  java.lang.NullPointerException
  java.lang.NullPointerException
      at NullPointerTest.main(NullPointerTest.java:5)

  End.
*/
```

Exceptions - IOTest.java I

```java
import java.io.File;
import java.io.FileReader;
import java.io.FileNotFoundException;

public class IOTest {

  public static void main(String[] args) {
    try {
      File file = new File("input.txt");
      FileReader fr = new FileReader(file);
    }
    catch (FileNotFoundException e) {
      System.out.println(e);
      e.printStackTrace();
    }

    System.out.println("\nEnd.");
  }
}
```

Exceptions - IOTest.java II

```
/* Output:
  $ java IOTest
  java.io.FileNotFoundException: input.txt (The system
      cannot find the file specified)
  java.io.FileNotFoundException: input.txt (The system
      cannot find the file specified)
      at java.io.FileInputStream.open0(Native Method)
      at java.io.FileInputStream.open(FileInputStream.java
          :195)
      at java.io.FileInputStream.<init>(FileInputStream.java
          :138)
      at java.io.FileReader.<init>(FileReader.java:72)
      at IOTest.main(IOTest.java:10)

  End.
*/
```

Exceptions - IndexBoundTest.java I

```java
public class IndexBoundTest {
  public static void main(String[] args) {
    int num[] = {1, 2, 3, 4};

    try {
      System.out.println(num[5]);
    }
    catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Exception thrown  :" + e);
      e.printStackTrace();
    }

    System.out.println("\nEnd.");
  }
}
```

Exceptions - IndexBoundTest.java II

```
/* Output:
  $ java IndexBoundTest
  Exception thrown  :java.lang.
      ArrayIndexOutOfBoundsException: 5
  java.lang.ArrayIndexOutOfBoundsException: 5
      at IndexBoundTest.main(IndexBoundTest.java:6)

  End.
*/
```

Exceptions - DivisionByZeroException.java I

https://docs.oracle.com/javase/8/docs/api/?java/lang/ArithmeticException.html

```java
public class DivisionByZeroException extends
    ArithmeticException {
}
```

Exceptions - MyDivision.java I

```java
public class MyDivision {

  public static double divide(double num, double denum)
      throws DivisionByZeroException {
    if(denum == 0) {
      throw new DivisionByZeroException();
    }

    return num/denum;
  }
}
```

Exceptions - MyDivisionTest.java I

```java
public class MyDivisionTest {

  public static void main(String[] args) {
    int num = 5;
    int denum = 0;

    try {
      MyDivision.divide(num, denum);
    }
    catch(DivisionByZeroException e) {
      System.out.println(e);
      e.printStackTrace();
    }

    System.out.println("\nEnd.");
  }
}
```

Exceptions - MyDivisionTest.java II

```
/* Output:
  $ java MyDivisionTest
  DivisionByZeroException
  DivisionByZeroException
      at MyDivision.divide(MyDivision.java:5)
      at MyDivisionTest.main(MyDivisionTest.java:8)

  End.
*/
```

Outline

1. Introduction
2. C++ to Java
3. Conclusion

Summary

- With respect to C++, Java simplifies several aspects in particular related to memory management.
- More elegant? Less flexible? (Too verbose?)
- Java arrived later: design had more time to learn from experience (e.g. generics introduced only in version 1.5 (relabeled as 5) in 2004).
- Arguably easier to transition from C++ to Java than the other way around.

What to do next?

Next Lab

- Lab assessment

Next Lecture

Week 10 - Revision [**May 2019 - Thursday 09/05 10:00-12:00** TBC]

Future: lifetime learning

- OO design: UML (other diagrams) & Design Patterns
- GUI (C++ Qt, Java Awt/Swing)
- Modern C++ (**C++11**, C++14, C++17)