Java is OO, widely used, powerful, flexible, portable, secure not the fastest language. Needs one or more class definitions.

OO is cool because objects can be used efficiently to represent real-world entities. Principles of OO are objects, attribute, method, class, encapsulation, inheritace, polymorphism.

INTERPRETER – runs the program

COMPLIER – needed for creating programs

DATA FIELD – named piece of data

METHODS – contain statements (steps) of programs

OBJECTS – instances of classes

ALGORITHM – description of processes

PRIMTIVE TYPE – float,long,byte,short,int,Boolean,char

REFERENCE TYPE – references to instance objects

VARIABLE – named bit of data. Declared & initialised

CLASS OBJECTS – one created automatically

INSTANCE OBJECTS - any number can be created (eg. Box box1 = new Box(x);)

APPLICATION – program 1 or more classes. One must be the application class (with main)

CLASS – declaration/header. Data fileds, methods ≤0

METHOD – decl/head. Local var, other statements ≤0

DATA FIELDS – used everywhere, specified visibility (need to be static if in class object), may be initialised

LOCAL VARIABLES – only used in their method, no need for visibility/static, must be initialised

JFC – java foundation classes (libraries, class libs, API)

API – application programming interface

GUI – Graphical User Interface

IDE – integrated development environment

AWT – abstract windowing toolkit

UMI – Unified Modelling Language

SUPPORT CLASS – non-static data fields and methods, instance objects made from these

APPLICATION CLASS – static data fields and methods

ENCAPSULATED – data protected by methods.

ACCESSOR – return the value in a data field. empty()

public String getName() { return name; }

MUTATOR – change value in a data field (void). (stuff)

public void setS(String s) { this.s=s; }

CONSTRUCTOR – used to create instances, must called. Don’t return type or void. Same name as class

CLASS MEMBER – static. In dashed boxes

INSTANCE MEMBER – not static. Solid boxes

MEMBER – data field or method

DOT NOTATION – test.y (data field). test.t() (method)

TOSTRING – every instance has toString(). Can replace with custom text to print

ALIASES – a reference to a variable

REFERENCES – point to classes, including library classes

SERVICE METHOD – called on an object (dotnot) public

SUPPORT METHOD – called by other methods within the class eg. Test.t(). private

STATEMENTS – declarations, executables

t\ - tab

n\ - new line

/\* \*/ = block comment

// = single line comment

/\*\* \*/ = javadoc comment

Syntax error = compile error (code is not legal)

Run time error = code compiles, impossible to execute code, when you run the program

Semantic error = no error messages, code compiles and runs, doesn't do what you expected

Boolean – 1 bit

Byte – 8 bit

Short, Char – 16 bit

Int, Float – 32 bit

Long, Double – 64 bit

fractions/decimals = float, double

Int (whole numbers) = byte, short, int, long

true/false = boolean

letters/digits/symbols = char

Strings are instance objects (reference type)

Import java.util.Scanner;

Scanner sc = new Scanner(System.in);

System.out.println(“some text here”);

String s = sc.nextLine();

+=, -=, \*=, /=, %=

I++ = evaluated to old then increments

++i = increments then evaluated to new

Math auto import with java.lang

Math.PI, Math.E, Math.sqrt(), Math.sin(), Math.max(), Math.pow(x,y)(x to the power of y)

widening ->

char, int, double

narrowing <-

double d = 1 (int conversion 1 to double 1.0)

double/int (int conversion to double. Result double)

int i = (int) 15.32 (cast double 15.32 to int 15)

Import javax.swing.\*;    - modern

Import java.awt.\*;    - old, builds on swing

Components = windows, frames, panels, buttons, text fields, scroll bars, check boxes, menus etc.

Java graphics are self contained (lightweight) and thus portable. Basic functions draw on the devices OS/windowing system (heavyweight).

All drawing code should be in a paintComponent (public void paint(Graphics g)

g.drawLine(x,y,x,y), g.setColor(Color.red)

Graphics are drawn with an origin at the top left (x co-ord, y co-ord) YAXIS IS UPSIDE DOWN

drawRect(int x, int y, int width, int height), fillRect same

drawOval(Int x, int y, int width, int height), fillOval same

drawArc(int x, int y, int width, int height, int startAngle, int arcAngle), fillArc colours pie slices

drawString(“Text”, int x, int y)

More recent drawings done over top

Import javax.swing.JFrame; (or javax.swing.\*;)

Public class Example0{

 Public static void main (String [] args){

    MyFrame obj1 = new MyFrame();

    obj1.setSize(int x, int y);

    obj1.setVisible(true);

    MyFrame obj2 = new MyFrame();

    obj2.setLocation(int x, int y);

Import javax.swing.\*; //need both for GUIs

Import javax.awt.\*;

Public class MyFrame extends JFrame{

 Public void paint(Graphics g) {

      ALL DRAW THINGS GO HERE

length(); returns an int

indexOf(); returns an int

substring(start index, end index);

replace(old char, new char);

charAt(index);

equals(“text”); use this instead of == for strings

import java.text.DecimalFormat;

DecimalFormat fmt = new DecimalFormat(“0.###”);

import java.text.NumberFormat;

NumberFormat fmt = new NumberFormat.getCurrencyInstance()

or .getPercentInstance();

fmt.Format(object);

WRAPPER – wrap a primitive value into an object. Integer obj = new Integer(40). Byte, Short, Integer, Long, Float, Double, Character, Boolean, Void (cannot be instantiated). byteValue(), doubleValue() etc. return the value of this Integer as the corresponding primitive type. num = Interger.parseInt(str); (change int into string).

AUTOBOXING = automatic conversion between a primitive value and corresponding wrapper object. Reverse is UNBOXING

True || true is true

True || false is true

False || false is false

True && true is true

True && false is false

False && false is false

Method call

!, -(unary), +(unary)

New, type cast

\*, /, %

+, -

<, <=, >, >=

==, !=

&&

||

=, +=, -=, /=, %=

if (condition){

statements;

} else if (condition){

statements;

} else {

statements;

}

switch(n){

   case 1:

          System.out.println(“something”);

          Break;

   case ‘a’:

        Stuff here

    default: (optional)

        Stuff

}

Have as many cases as you like, use breaks to end the switch after a case if you want

while(condition){ //only does if passes conditions

statements;

}

do { //always does at least once

  statements;

} while (condition);

String [ ] words = {“The”, “quick” etc };

for (String s : words){

   S.out.println(s); //will print everything in ‘words’

}

char c = ‘a’;

for(initialisation, condition, update)

 statement

Eg. for(int = 2, i <= 7, i++)

for(type var: collection){}

words[1] will access the word at that index in the list

private data fields enforce encapsulation. Methods should be public for service methods, private for support methods

p1 = new Point(7,4);

p2 = p1;

p3 = new Point(7,4);

p1 == p2 is true – aliases/references are equal

p1 == p3 is false – but p1.equals(p3) is true

this always refers to the current (instance) object.

Private int x,y;

Public Point(x,y){

this.x = x;

this.y = y;

}

An array is a named collection of data of the same type and fixed size. Index starts from 0. Is an object. Can hold references to objects

int [] array1 = new int [5]; - 5 is the size of the array

for(int I = 0; i <array1.length; i++){

x[i] = readInt(“Enter an int for cell” + i);

} // how to fill an array with user entered ints

int [] x = {4,7};

a.length – length of an ARRAY called a

str.length(); - length of a STRING called str

int [] [] a = new int [3] [5]; - 3 down, 5 across

a[2][3] will access the element row 2 column 3

int [] [] a = {{2,3,4}{5,6,7}}

print a 2D array called b

for (int row = 0; row<b.length;row++){

for (int col = 0; col < b[row].length; col++){

System.out.print(b[row][col] + “ “);

}

System.out.println();

}

or

for (double[] row : b){

for (double x : row){

System.out.print(x + “ “);

}

System.out.println();

}

array a = b, c is the same nums as a/b

a == b is true

a == c is false

Arrays.equals(a,b) is true

Arrays.equals(a,c) is true

Main explained

public class Demo{

public static void main (String [] args){

System.out.println(args[0] + “\*\*\*” + args[1]);

}

}

GUI  
COMPONENTS – objects drawn in the GUI eg buttons, also containers like panels and frames

EVENTS – objects that represent events eg button clicks

LISTENERS – objects that get sent events and have methods set up to process them

FRAME – a container that is drawn as a separate window. Heavyweight component. Several layers of panes, main one is content pane

PANEL – container can only exist as a subpart of another container (eg frame). Hold and organise others components. Lightweight component

Application class always has these 5 steps

1. Make frame/window
2. Define close behaviour
3. Add support panel
4. Auto size the frame (pack)
5. Make frame visible

Support classes always have these 3 steps

1. Define components
2. Add components to panel
3. Set panel details (eg size)

FlowLayout – left to right then top to bottom. Default

BorderLayout – north south east west center

BoxLayout – in a single row of column

GridLayout – within cells of a grid with standard sized rows and columns

add(b1, BorderLayout.SOUTH);

Containment Hierachies

1. Create components
2. Add components to panels
3. Add inner panels to the main/outer panel
4. Add the outer panel to the frame

INNER CLASS – declared inside another class. Only the outer can see/create instances of the inner. Members of outer are in scope for inner. Feature for safety and encapsulation

INTERFACE – like a restricted form of a class. Contain only abstract methods and static final data fields. Other classes may implement 1 or more interfaces. Enforce good design. Eg. implements WindowListener, ActionListener

event.getSource(), label.setText(“text”), s.getText(), e.isSelected(),

CASTING – most general form of conversion in java. dollars = (int) money; result = (float) total/count;

Timer t = new Timer(1000, new MyListen());

t.start(); and t.stop(). 1000 is milleseconds, MyListen is the action listener method name

STREAM – is a source or destination of a sequence of data. Used as an abstraction for dealing uniformly with files and devices

TOKENS – text items separated by “white space” – blanks or tabs

Scanners use tokens. Scanner sc – new Scanner(“axe bag cat”);

System.out.println(sc.next()); etc for each token OR

while (sc.hasNext()){

System.out.println(sc.next());

}

also nextInt(), nextDouble(), nextLine()

try{

// code that might cause an exception

} catch (ExceptionClass variable){

//warn the user to try to handle the exception

}

example exceptions. /0 = ArithmeticException, IOException (file read fail), InputMismatchException (unexpected data type)

outfile.close(); to close a file after you write it

reading from file – import java.io.\*;

Sorting and Searching are main components of Comp Sci

BUBBLE SORT – swapping smaller neighbours with bigger ones

SELECTION SORT – keep putting the smallest thing at the start

MERGE SORT – split into two sub-lists, sort them, merge two lists

INSERTION SORT – take the next thing and put in order with past things

Selection sort has the same characteristics no matter which language or machine

EXTENDS – can extend almost any existing class to create/define a new class. A child class has its own members and also the non private members of its parent class

By default, every class extends java.lang.Object. toString is inherited from Object

public Person(String name, int age){

this.name = name;

this.age = age;

}

public Person(String name, int age, int height){

this(name,age);

this.height = height;

}

this() and super() have to be the first line in a method. Have one or the other but not both

SUPER = is a keyword referring to members of the current class/object inherited from the parent

CONSTRUCTOR CHAINING – every object is constructed by calling its own constructor, which calls a (maybe automatically) parent constructor, which calls a parent etc. up hierarchy to object constructor so objects are properly initialised

PACKAGES – group related classes together

CLASSES/VISIBILITY – set visibility of data fields. Abstraction. Encapsulation. Information hiding

METHODS/SCOPE – organise processing. Variables only within a method

BLOCKS {}/SCOPE – Organise processing. Can define local variables. In blocks within methods (scopes)

Packages and classes are for large programs/teams. Classes, methods, blocks are for small programs/individuals

Every class is part of a package. Eg java.lang. Classes you write/put in a package are in the default package

PACKAGES support OO concepts in java, involve some practical details about managing classes and files, define a “namespace” for the classes they contain. Classes in package must be stored in a single directory which should have the same name as the package

package java.lang; OR package com.sun.name;

import package.name;

CLASSPATH – tells compiler what directories to search

NAMESPACE – use fully qualified name eg java.util.Random or simple name (Random) if the class is in the same package or your source files import the other classes (java.util.\*) or the other class is in java.lang

Importing a packages means we don’t have to type out the full name every time we want to use the class

Visibility

PRIVATE – accessible within only this class

PACKAGE – accessible only to classes in the package. Doesn’t particularly have a name because default. Promotes software reuse

PROTECTED – accessible to classes in this package and any classes extending this one

PUBLIC – accessible to all other classes (if class public)

ADT – abstract data type. All internal structor is hidden from the user. Can use classes as ADT by allowing access via public methods

INTERFACE – public methods of a class. Only interface needs to be understood. Everything else is hidden/safe (encapsulation/info hiding/ADT)

METHOD OVERLOADING – classes can have multiple methods with the same name as long as they have different signatures. Makes the structure of a program clear. If no matching signature it’s a compiler method

METHOD OVERRIDING – when methods with the same signature exist in a parent class and a child class

BOTH PROMOTE COHESION

Overloading is shooting 3 arrows at once

Overriding is shooting 2 arrows taped together (long)

When you rewrite a method with the same signature in a child class the parent class becomes overridden and can only be accessed with super.methodName(); Same with SHADOWING (datafields with the same name)system.out.print (childClassName.variable); also

instanceOf – operator which returns true if an object is an instance of a class eq (s instanceOf Square)

Methods with the same name can exist in the same class with different signatures (overloading), a class and parent or other superclass with same signature (overriding), a class and its parent with different signatures (weak polymorphism)

Members are instance members (part of instance objects) unless the modifier STATIC specifies they are class members. Class data fields are shared by/have the same value for all instances of a class. Class methods can only access class data. Class members already exist as part of a class object when a program starts running. They are independent of any instance objects.

POLYMOPRHISM – ability of objects belonging to different types to respond to method calls of methods of the same name, each one according to an appropriate type-specific behaviour. Programmer and program don’t have to know the exact type of the object in advance, so this behaviour can be implemented at run time (late binding or dynamic binding). Eg overloading, overriding

ABSTRACT CLASSES – used when you want to create instances of subclasses but NOT the abstract parent/ subclass class itself eg abstract Shapes because only create instances of Circles, Squares etc. Similar to interfaces in that you can treat subclasses as type Shape, ensure every method od type Shape will have certain methods and data fields. Different from interfaces in that you extend only one AC, but can implement many interfaces, and an AC can only have its normal methods (data fields don’t need to be public static final)

Public abstract class Shape {

public abstract void showArea();

}

public class Circle extends Shape{

public void showArea(){

stuff here no in the last class

}}

Shapes [] can nclide circles, squares etc in AC

MULTIPLE INHERITANCE – eg extends x,y not allowed

FINAL – final class cannot be extended, overridden, or change data field once initialised and declared. Can be instantiated

ENUMERATED TYPES – are a way of creating a new type by enumerating (listing) every possible value

Enum Fruit {apple, orange, banana, cherry, grape}

Public static void main (String [] args){

Fruit f = Fruit.apple;

}

f.ordinal() – returns ordinal position of f (from 0)

f.name() - returns value/name of f

for (Fruit fru : Fruit.values()){

s.o.p(“a nice “ + fru);

}

GENERICS - programs can be written in terms of “to be specified later” types that are instantiated when needed for specific types (provided as parameters). Before explicit casts were needed now we can <v>

import java.util.ArrayList;

ArrayList a = new ArrayList();

a.add(“Hello”);

S.O.P(((String)a.get(0)).length()); //old

ArrayList<String> b = new ArrayList<String>();

b.add(“Hello”);

S.O.P(b.get(0).length()); //new

COLLECTION API – includes ArrayList, copy(), min(), max(), fill(), reverse(), shuffle(), sort(), contains()

ARRAYLIST – can hold objects, don’t need to declare size, can grow and shrink as required. Support generics. Can’t store primitive types unless wrapper

In an ArrayList a.get(var/int), a.indexOf(“name”), a.add(“name”), a.add(index,”name”), a.remove(index)

If no type is specified, the ArrayList stores objects as type object. All below are allowed but it is risky. Depend on complier/IDE settings may get a warning

int [] x = {8,7,6}

ArrayList al = new ArrayList();

al.add(“Hello”);

al.add(new MyClass(42));

al.add(x);

to use objects in such a list must cast them to type

String s = (String)al.get(0);

((MyClass)al.get(1)).myMethod();

APPLETS – are not run directly like applications, but by and within another program called an applet context. Eg. Web browsers and applet viewers. They cannot access files on the machine. Run an applet by loading an html file into a browser. Browser calls the init method

GOOD DESIGN – packages, visibility, scope, inner classes, type checking, array bounds checking, references instead of pointers, garbage collection instead of memory allocation and deallocation

SIMULATION – is an attempt to model a real life situation on a computer so that it can be studied to see how the system works. By changing variables, predictions may be made about the behaviour of the system

1. Solve the problem
2. Organise your program well
3. Keep your data safe
4. Write a big problem in small pieces
5. A working program may not be a good program
6. Anticipate errors
7. Start again instead of patching bad code
8. Clear code is better than clever code
9. Good formatting and commenting
10. Read the friendly manual (RTFM)

ENCAPSULATION – the characteristic of an object that means it protects and manages its own information

SEMANTICS – rules that define what a statement in a language means

INSTANTIATION – process of creating a new object

POLYMORPHISM – the ability to define an operation that has more than one meaning by having the operation dynamically bound to methods of various objects























