Computer Science 3A - CSC3A10/CSC03A3

Lecture 1: Java Basics and OO

Academy of Computer Science and Software Engineering
University of Johannesburg

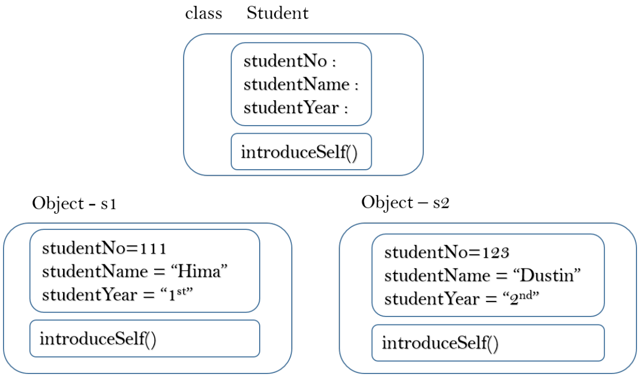
- Exceptions
- Casting
- Generics
- Exercises

Java Programming

Classes and Objects

- Class
  - Blueprint
  - Properties and Methods
- Objects
  - primary ”actors”
  - new, Dot operator

Class and Objects



Classes, Types and Objects

- Class modifiers (public, abstract, final)
- Variable modifiers (public, protected, private, static, final)
- Enums

```
enum Day{
   Monday,
   Tuesday,
   ...,
   Sunday
};
Day x = Day.Monday;
```

Methods

Method modifiers:

- public
- protected
- private
- static
- abstract
- final

Method Types

Method types:

- Procedure
- Function
- Constructor
- Main

```
public static void main (String[] args){
//Main code here
}
```

Main method

Expressions

- Literals (datatype variable = literal;)
- Operators
  Assignment, Arithmetic, String Concat(+),
  Increment and decrement (eg. `++i`), logical, bitwise
- Operator precedence
- Casting (explicit cast and implicit cast)

Control Flow

- If statement
- elseif
- switch
- loops (`while`, `for` and `do-while`)
- Explicit control-flow statements (`return`, `break` and `continue`)

Java Packages

- Packages
  built-in, user-defined
- built-in packages
  import java.util.Scanner;
- user-defined
  source code located under dir "packageName"
  each file begins with "package packageName;"

General Java programs

Developing a Java program:

- Design
- Psuedo-code
- Coding
- Documentation
- Readability and Style
- Testing
- Debugging

Exercises

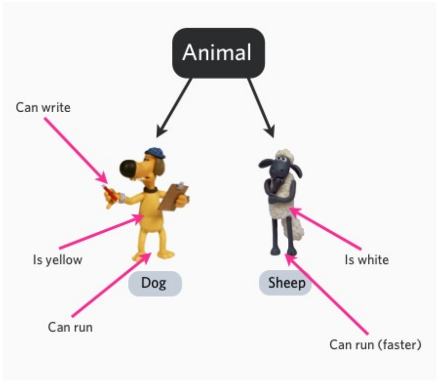
Reinforcement exercises:

- R1.2
- R1.3
- R1.4
- R1.8

Creativity exercises:

- C1.10
- C1.12
- C1.13
- C1.17

Object Orientation



OO Goals

- Robustness
- Adaptability
- Reusability

OO Design principles

- Abstraction
  - Distill complex system into its most fundamental parts
  - Applying abstraction paradigm - Abstract data types (ADT)
- Encapsulation
- Modularity
- Hierarchical organization

Design Patterns

- Template for a software solution
- Consists of name and context (describes scenario for usage)
- Algorithm patterns
- Software engineering patterns

Inheritance

- Modular and hierarchical organization structure
- Base class or super class
- Subclass inherits (extends) the base class
- Dynamic dispatch/binding

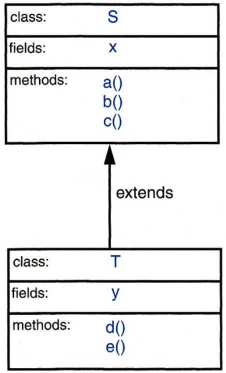


Figure: Inheritance example, where S is the parent of T

Polymorphism

- "Many forms"
- In OO design, objects take different form
- Override
- Overloading (with a different signature) or name, type and argument combination)
- **Self study** - Using inheritance in Java and numeric progression example

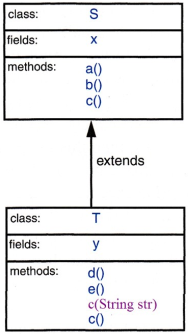


Figure: Polymorphism example, where the c method has multiple definitions

Interfaces

- Method declarations with no body and no data
- Methods are always empty
- May not be instantiated
- Class implementing interface must implement all interface's methods

Interfaces II

```
public interface Sellable
    {
        public string description();
        public int listPrice();
        public int lowestPtice();
    }

public class Product implements Sellable
        //...
        public string description(){ return description;}
        public int listPrice(){ return price;}
        public int lowestPtice(){ return price*0.5;}
    }
```

Interface example

Multiple Inheritance

- No multiple inheritance for classes allowed
- Multiple inheritance on interfaces is allowed

```
public class MotorCar extends LandVehicle implements ISellable,
    IPurchasable
```

Multiple Inheritance alternative

Abstract class

- Empty method declarations
- Concrete declarations of methods and variables
- May not be instantiated
- Can extend other abstract classes

```
public abstract class Number{...}
```

Abstract class example

Exceptions

Throwing exceptions:

- Objects that are thrown when unexpected condition experienced
- Thrown exceptions are caught

```
if (some condition)
  throw new MyException(''We have a problem!'');
```

Exceptions II

Throw clause specifies throw exceptions at declaration

```java
public void watchingRugby() throws noBeersException, noBiltongException

public void PlanningSaturday() throws noBeersException,
    noBiltongException
{
  watchingRugby() ; //do not have to have try catch block
}
```

Throws example

Exceptions III

Catching exceptions require a try-catch block

```
try
{
    PlanningSaturday();
}
catch (Exception e){
    if (e instanceof noBeersException)
      sendFriend();
}
finally startBraai;
```

Throws example

Exceptions IV

finally

- optional
- Executed regardless of exceptions being thrown or caught

Casting

- Casting up
- *java.lang.Object ⇒ java.lang.Number ⇒ java.lang.Integer*

```
//Integer i= new Integer(3); Deprecated with Java 9
Integer i = Integer.valueOf(3);
Number n = i;
```

Casting up

Casting II

- Casting down

```
Number n = Integer.valueOf(2);
Integer i = (Integer) n;
```

Casting down

Casting III

Casting exceptions

- **instanceof**

```
if (n instanceof Integer)
  Integer i = (Integer) n;
```

Multiple Inheritance alternative

Casting IV

Casting with interfaces

```
public interface Person {
    public boolean equalTo (Person other);
}

public class Student implements Person {
    //...
    public boolean equalTo (Person other){
        Student otherStudent = (Student) other;
        //...
    }
    //...
}
```

Casting with interfaces example

Generics

Generic framework

- Abstract types that avoid many explicit casts
- Define a class in terms of formal type parameters

```
public class Pair<K,V> {...}
```

Single-letter uppercase names

Generics II

Generic type

- not defined at compile time
- specified at run time
- Instantiate an object with actual type parameters

```java
Pair<String, ArrayList> = new Pair<String, ArrayList>();
```

Generic instantiation

Generics III

Generics can also be restricted

```
public class Pair<K,V extends ArrayList> {...}
```

Generic instantiation

Exercises

Reinforcement exercises:

- R2.3
- R2.4
- R2.5
- R2.6
- R2.9
- R2.10

Creativity exercises:

- C2.12
- C2.14
- C2.18