[**Java concepts**](#_muhhf7ver4rp) **2**

[**Constructor**](#_188hxxs3fy2d) **2**

[**Inheritance**](#_ursvmqbiyodt) **2**

[**Static**](#_5gi93ot7255n) **2**

[**Interface**](#_d0lo0jpxk3j3) **3**

[**Abstract classes**](#_rivvh8sysq4k) **3**

[**UML**](#_iwfhybeh9c7g) **4**

[**Generic classes**](#_l43tdc5zb06y) **5**

[**Coding methods**](#_ltxi4eeszxxy) **7**

[**Exception and Errors**](#_et8x78qa98nj) **7**

[**Javadoc**](#_466dodwxhz1z) **10**

[**JUnit Software testing**](#_6l9gxe4hhyab) **11**

[**Event-Driven Programming**](#_pm1hpecqrn27) **12**

[**Advanced java concepts**](#_rx159irzbjy7) **13**

[enum](#_ai71itwjbt9q) 13

[Variadic Method](#_jxb4vytr9pwd) 14

[Functional Interface](#_vsyap2ppabnn) 14

[Lambda Expression](#_e8gyou3v0frd) 14

[Method reference](#_rp0wqh624bw) 16

[Stream](#_v8qdh2mdxovr) 16

[**Other things**](#_ayfij7w04hjk) **17**

[**Shortcuts**](#_slwf27yhxfv9) **18**

# 

# 

# **Java concepts**

**Encapsulation**: The ability to group data (attributes) and methods that manipulate the data to a single entity through defining a class is called encapsulation

**Polymorphism**: The ability to use objects or methods in many different forms. Overriding, overloading, substitution (subclasses), generics

**Data Abstraction**: Using abstract classes and methods

**Information hiding**: Private, protected

**Delegation**: Delegation can be viewed as a relationship between objects where one object forwards certain method calls to another object, called its delegate.

**Inheritance**: Subclass

# **Constructor**

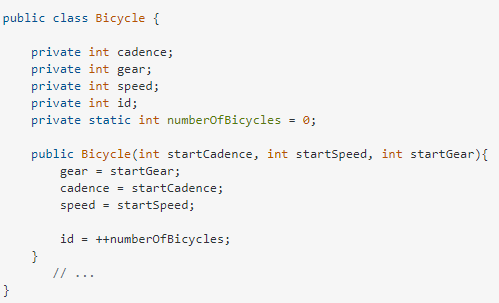
* A constructor can initialize attributes defined as constants within the class only if they are not pre-initialized.

# **Inheritance**

* Override statement is optional - purely for readability
* Override cannot change return type
* Cannot override private methods only public and protected
* super() has to be the first line in the constructor if it is called.
* Calling subclass from parent example:

# **Static**

* A variable that is shared among all objects of the class; a single instance is shared among classes. Such an attribute is accessed using the class name.
* Static variables: One copy per class/ shared across all objects. e.g. numCircles (total number of circle objects created)
* Static methods cannot make use of the this and super keywords.
* Static methods can only call other static methods.
* Static methods can only access static data.
* Can use static attributes in non static methods.
* IF called outside the class, static attributes should be called by: classname.attributename



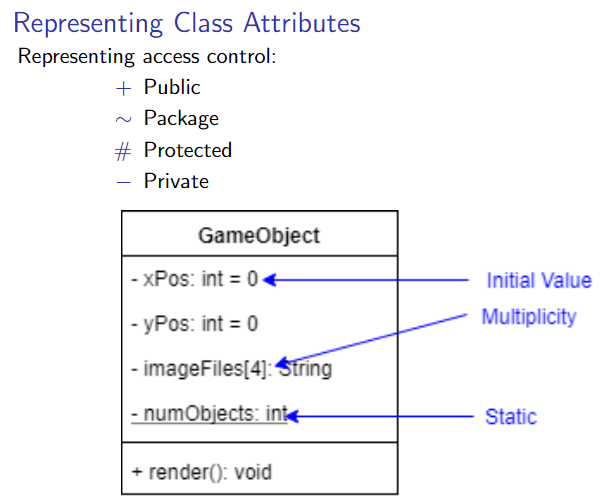
# **Interface**

* Only 1 constructor
* Implies a "can do" relationship. eg, circle can be drawable
* A class can implement multiple interface but only extend 1 class.
* Can implement methods if Default is used
* Classes that implements interface must override all methods (Unless Default keyword)
* All methods are implied to be abstract and public and all attributes are public, static and final variables by default
* A class that does not implement all methods in an interface must be abstract.
* An interface can extend other interfaces, just as a class subclass or extend another class.
* Implies a "can do" relationship. eg, circle can be drawable

# **Abstract classes**

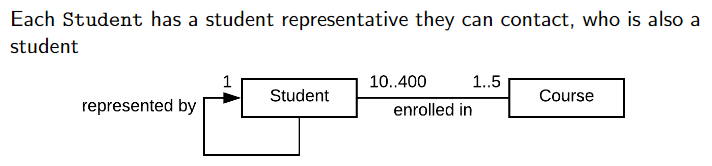
* Implies a "is a" relationship. eg, circle is a shape
* classes can only extend one class
* Abstract class can have both abstract and concrete methods
* Abstract class can have final, non-final, static and non-static variables.
* Classes that extends abstract classes may not have to override all methods
* Subclass must have to implement all the abstract methods of abstract parent class

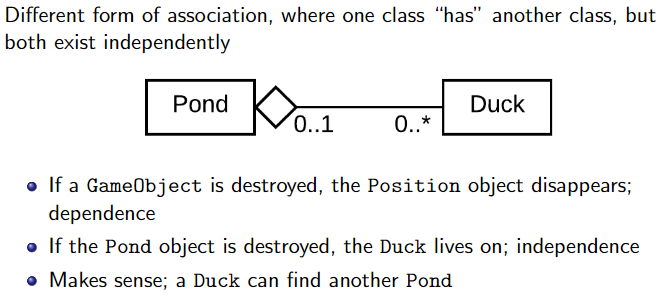
# **UML**

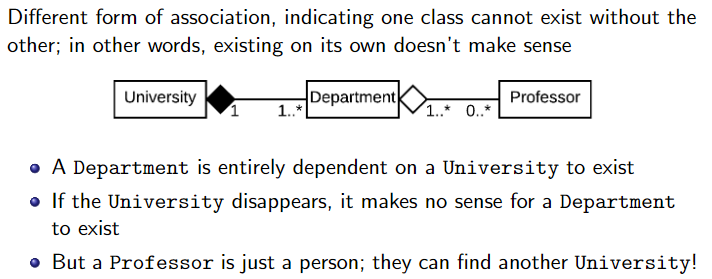


Static is underlined

Italic class/methods means abstract







# **Generic classes**

Same methods but different attributes

public interface Comparable<T> {

public int compareTo(T other);

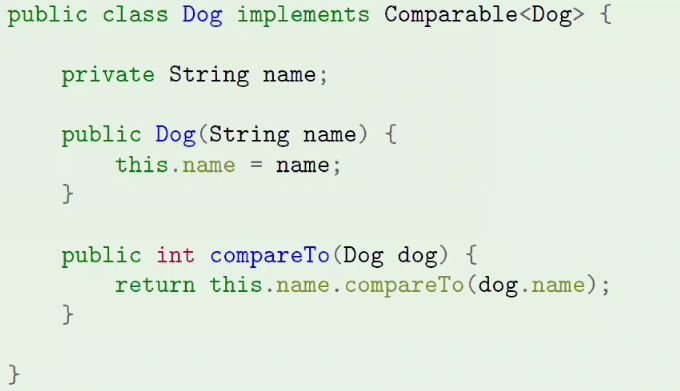
}

T is a type parameter, or type variable When T is given a value (type), every instance of the placeholder variable is

replaced

The value of T is literally a type (class/interface);Integer,String,Robot,Book,Driveable

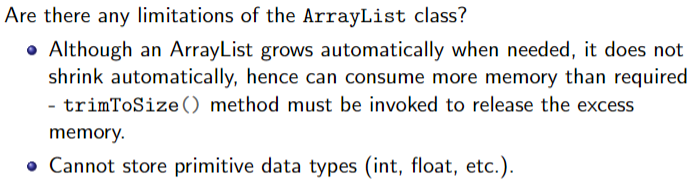
Whoever is implementing the interface must provide the type.



If we use the non generic object argument for comparable, if we compare two different objects (circle and square)

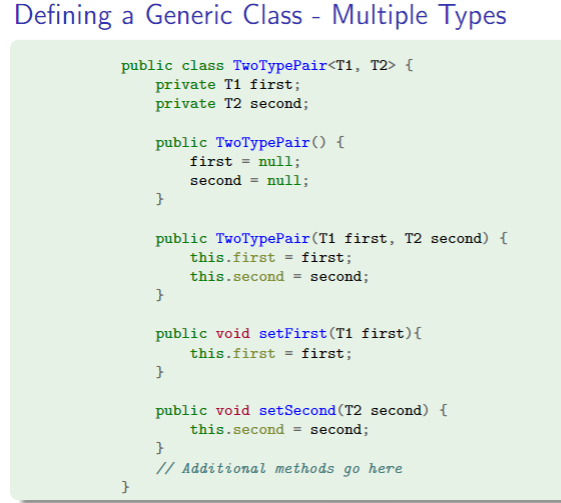
it would not give us a compilation error. Which is bad because it may cause errors next time.

Thus we use a generic class so that if we compare two different objects, an error would occur on compilation.



Can put different objects in an ArrayList

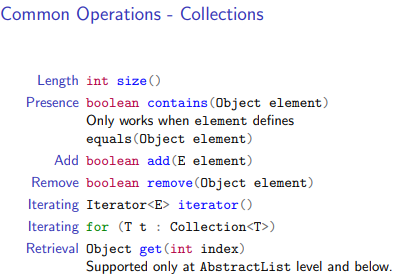
new ArrayList<Object>();



public class Rectangle<T extends Number> //can extend class or interface

Number is a parent class of Double, Long, Integer

public void addAll(Collection<? extends T> collection)



Sorting arraylist:

need to have a compareTo method to determine how it is sorted. Returns an integer

function: Collections.sort(arrayListVariable)

if it is a string, we can use compareto method to sort string.

Each of these have their useful applications

* ArrayList: like arrays, but better
* HashSet: ensures elements are unique - no duplicates
* PriorityQueue: allows you to order elements in non-trivial ways
* TreeSet: Fast lookup/search of unique elements

# **Coding methods**

* The Singleton Design pattern is used to ensure there is a single global instance of an object. Like global variables in C, it is usually best to avoid this pattern.
* The Factory method design pattern allows more flexible control over which specific instances are created e.g. of an abstract parent class or interface.
* The Template method design pattern allows an implementation of an algorithm to vary in its details using inheritance.
* The Strategy Design pattern allows an implementation of an algorithm to vary in its details using delegation.
* The Observer Design pattern decouples class interactions by notifying an observer when a subject’s state changes.

# **Exception and Errors**

There are 3 types of errors:

* Syntax:Errors where what you write isn’t legal code; identified by the editor/compiler.
* Semantic:Code runs to completion, but results in incorrect output/operation;

identified through software testing

* Runtime:An error that causes your program to end prematurely (crash and burn);

identified through execution.

Common runtime error:

* Dividing a number by zero
* Accessing an element that is out of bounds of an array.
* Trying to store incompatible data elements.
* Using negative value as array size
* Trying to convert from string data to another type
* Opening a file in read mode that does not exist or no read permission

Exception: An Error state created by a runtime error in your code; an exception.

Exception: An object created by Java to represent the error that was encountered.

Exception Handling: Code that actively protects your program in the case of exceptions.

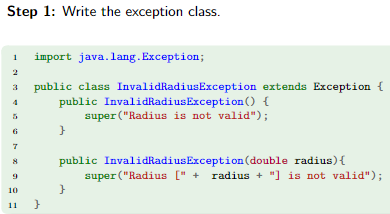
An exception is an event, and the catch block is an event handler.

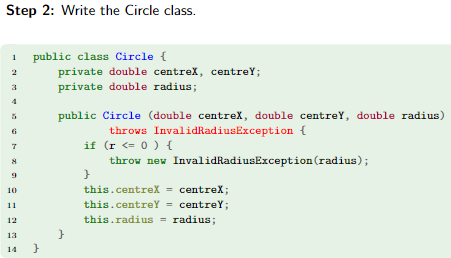
The keyword throw is used to generate an exception

throw:Respond to an error state by creating an exception object, either already existing or one defined by you.

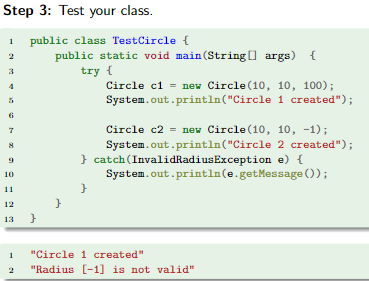
throws:Indicates a method has the potential to create an exception, and can't be bothered to deal with it, or that the exact response varies by application.

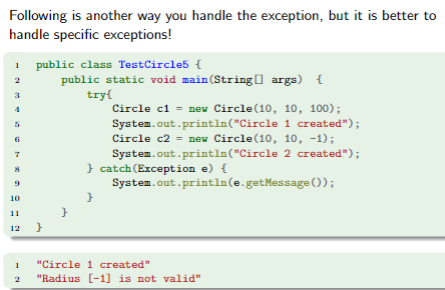
Creating own exception





note this is “throws”





Exception has to go from more general to more specific

Exceptions types:

Unchecked:

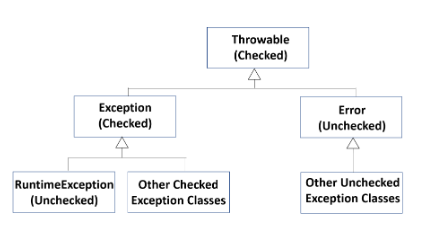
Can be safely ignored by the programmer; most (inbuilt) Java Exceptions are unchecked,

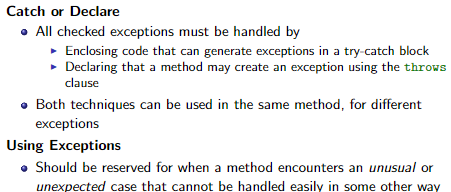
because you aren’t forced to protect against them.

Checked:

Must be explicitly handled by the programmer in some way; the compiler gives an error if a checked exception

is ignored.



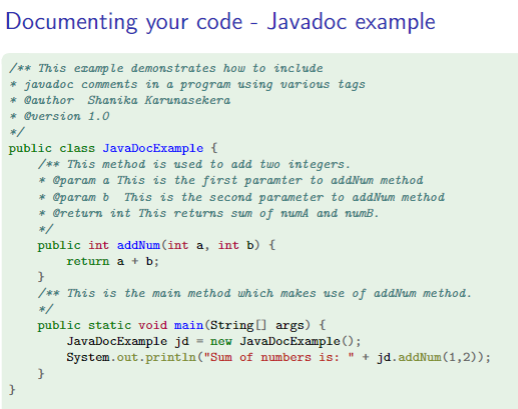


# **Javadoc**

Automatically generates a documentation based on your comments in your code using

@param

@return

Design principles

* Modularity: Decomposing the problem to units (modules) that are easy to understand, manage and re-use.
* Cohesion: Modules must be designed to solve clear, focused problems. Designs must have high cohesion.
* Coupling:The degree of interaction between modules must be reduced as much as possible.

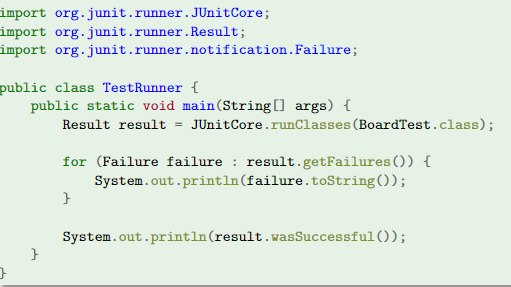
Designs should have low coupling.

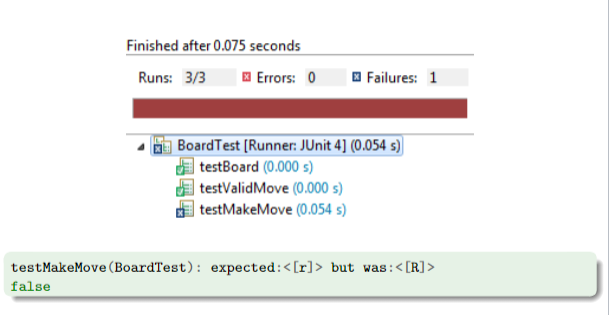
* Open-Closed Principle:Modules should be open to extension, but closed to modification.

# **JUnit Software testing**

* TestCase class:A class dedicated to testing a single unit.
* TestRunner class:A class dedicated to executing the tests on a unit.
* assert:A true or false statement that indicates the success or failure of a test case.









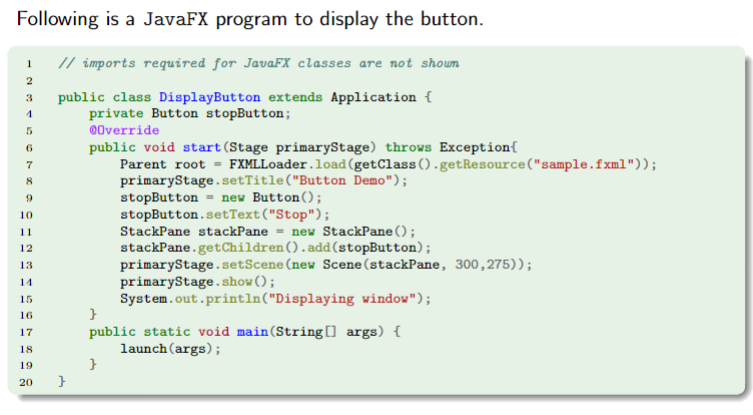
# **Event-Driven Programming**

Using Events And Callbacks To control the flow of a program's execution based on changes to the program state.

State: The properties that define an object or device; for example, whether it is“active”.

Event: Created when the state of an object/device/etc. is altered.

Callback/Listener: A method triggered by an event.



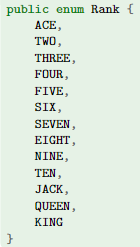


# **Advanced java concepts**

## **enum**

A class that consists of a finite set of constants

Must list all values.



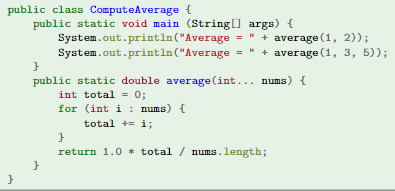
sort will sort based on enum



## **Variadic Method**

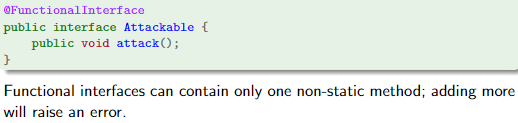
A method that takes an unknown number of arguments.

Variadic methods implicitly convert the input arguments into an array.



## **Functional Interface**

An interface that contains only a single abstractmethod; also called a Single Abstract Method interface.

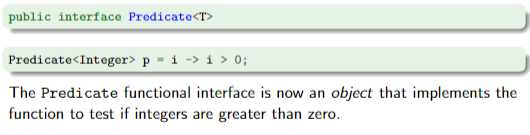


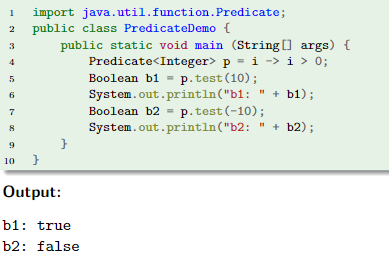
## **Lambda Expression**

A technique that treats code as data that can be used as an “object”;

for example, allows us to instantiate an interface without implementing it.

The lambda expression applies one method to every element of the list.

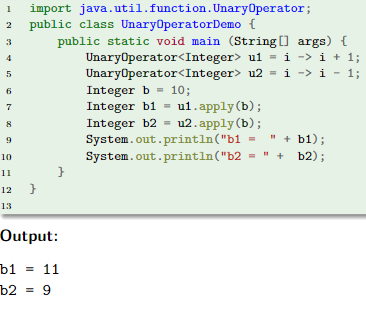




A lambda expression takes zero or more arguments (source variables) and applies an operation to them

(sourceVariable1, sourceVariable2, ...)

-> <operation on source variables>

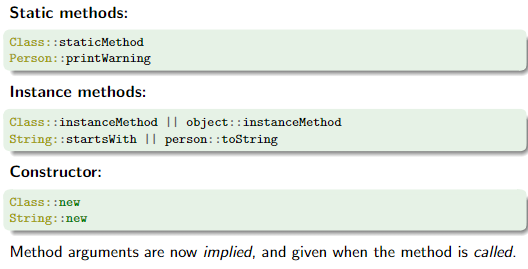


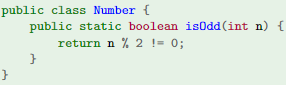
## **Method reference**

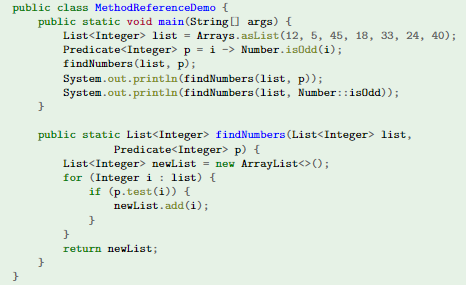
An object that stores a method; can take the place of a lambda expression if that lambda expression is only used

to call a single method.



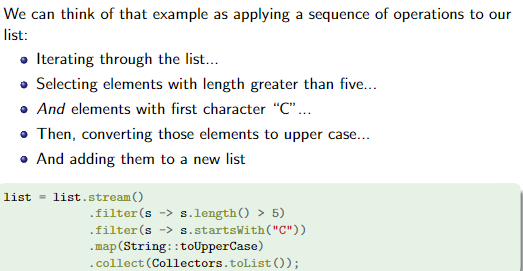






## **Stream**

A series of elements given in sequence, that are automatically put through a pipeline of operations.





# **Other things**

The == operator checks for reference equality, while the equals method is used to check for logical equality.

Upcasting: When an object of child class is assigned to a variable of an ancestor class -

Piece p = new Rook(2,3)

Downcasting: When an object of an ancestor class is assigned to a variable of a child class -

Robot robot = new WingedRobot();

WingedRobot plan = (WingedRobot) robot;

To call a subclass method from the superclass, you have to declare the method in the superclass. Then override it in the subclass. The method body from the subclass will be invoked when the superclass calls it.

protected ArrayList pipeList; //can also do ArrayList<AbstractPipeSet>

pipeList = new ArrayList();

((AbstractPipeSet) pipeList.get(i)).update();

HashMap<Character, Integer> result = new HashMap<>();

# **Shortcuts**

alt insert for getters and setters

control O for override

psvm for main

sout for System.out.println()

psf for private static final

control Q for description of contents

**Exam prep:**

Open intellij

Open merged slides

Open this notes