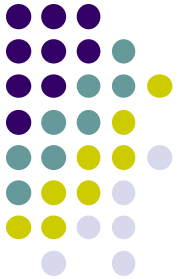


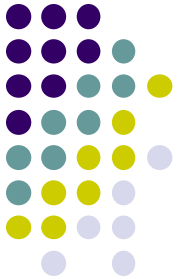
JAVA

Naveen Kumar K S
Naveen@probits.in

Session Plan

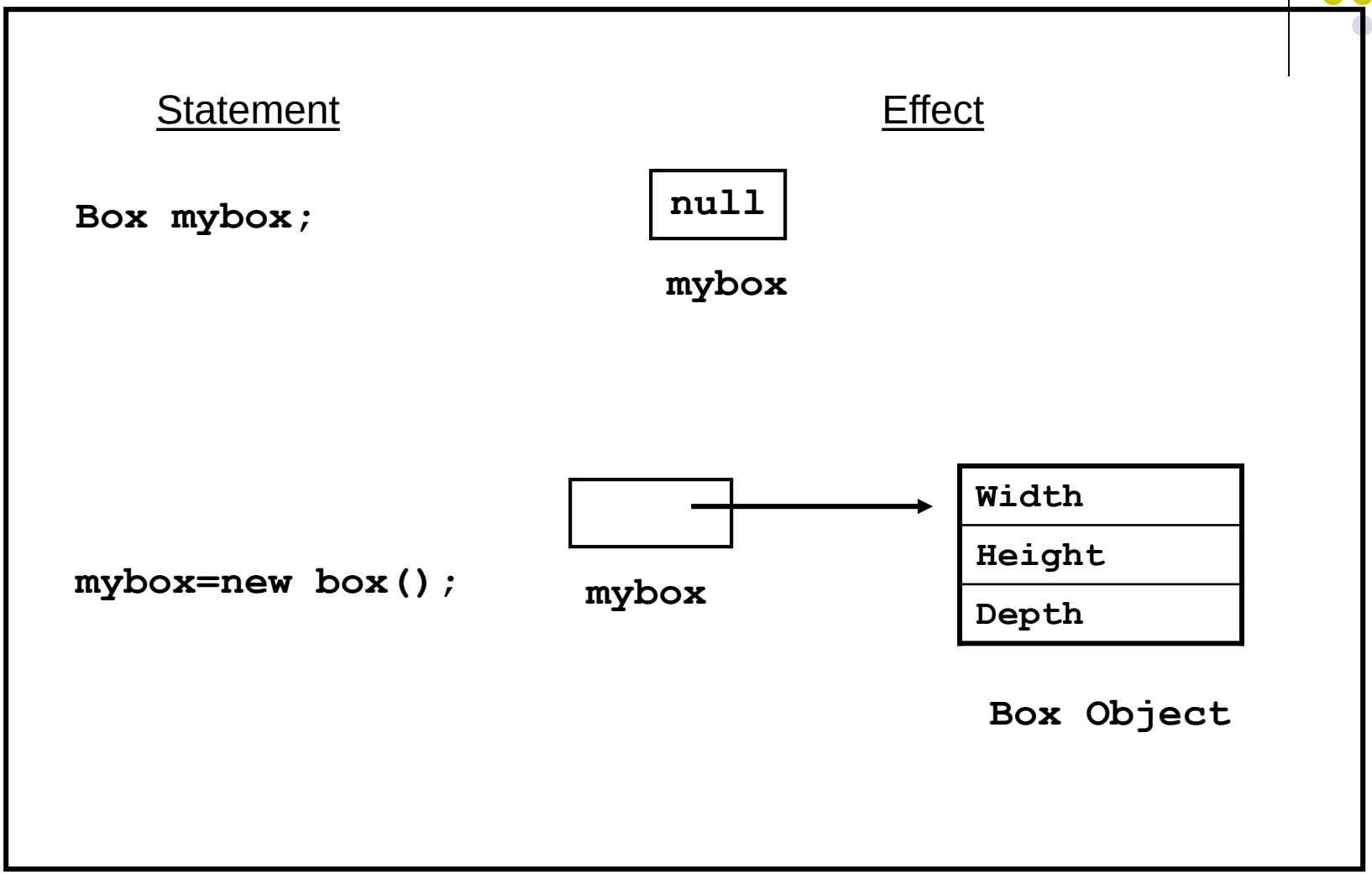
- The basic constructs in Java
  - ✓ To introduce Classes, Objects & Methods
  - ✓ To introduce inheritance
  - ✓ Packages & Interfaces
  - ✓ Exception Handling

Classes in Java



- The class is at the core of Java
- A class is the combination of Instance variables and Methods
- The data, or variables, defined within a **class** are called *instance variables.*
- General form of java does not specify a main() method.
- The **new** operator dynamically allocates memory for an object.

Declaring an object of type Box

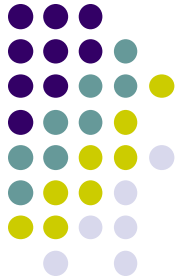


Introducing Methods

```
type name(parameter-list){
        //body of method
}
```

- Here type specifies the type of data returned by the method.
- If the method does not return a value its return type is **void**.
- Parameter list is separated by commas.

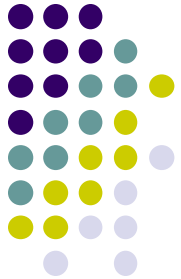
Constituents of a Class



- Constructors
  - ✓ Called immediately after the object is created before the **new** operator completes
  - ✓ They have no return type, not even **void**
- **this** Keyword
  - ✓ Used inside method to refer to the *current* object
  - ✓ Instance Variable Hiding
- Garbage Collection
  - ✓ No explicit need to destroy objects as in C++
- **finalize()** Method
  - ✓ An Object will need to be perform some action when it is destroyed.

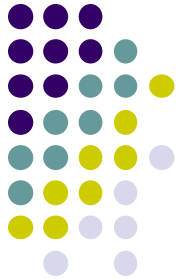
    Ex: Some non-Java resource such as file handle or window character font

Overloading Methods



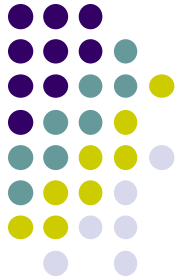
- Two or more methods within the same class that share the same name
- It is one way of implementing Polymorphism
- It is most exiting & useful Features
- Overloaded method is invoked using type and/or number of arguments

Access Control



- You can control what parts of a program can access the members of class
- You can prevent misuse
- Access specifiers
  - ✓ **public**
  - ✓ **private**
  - ✓ **protected**
- **protected** applies only when inheritance is involved

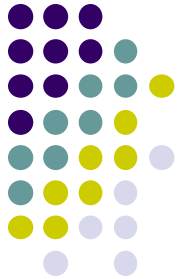
Public-Private



- **Public** variables and methods are those which can be accessed from any where i.e.
  - ✓ from the class
  - ✓ outside the class
  - ✓ outside the package

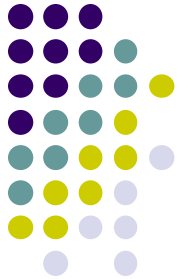
- **Private** variables are those which can be accessed only within the class.
  - ✓ They are not visible outside that class

Protected



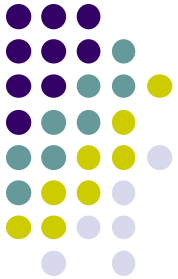
- **Protected variables**
  - ✓ visible only inside the class and the children classes of that class.
  - ✓ If a class extends a base class then the derived class will be able to access the variables and methods of the base class that are declared as protected (and public of course….)
  - ✓ Visible inside the package also

Default Scope



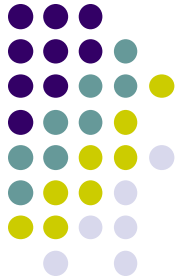
- The **default** Scope
  - ✓ package scope.
  - ✓ within the package, the class will be accessible but outside the package it is not accessible.

Class Member Access



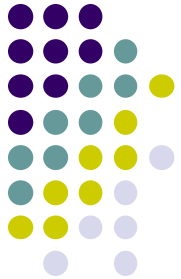
	Private	Package	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same Package subclass	No	Yes	Yes	Yes
Same Package non-subclass	No	Yes	Yes	Yes
Different Package subclass	No	No	Yes	Yes
Different Package non-subclass	No	No	No	Yes

Understanding **static**



- It is accessed before any objects of its class are created & without reference to any object
- You can declare both methods and variables as static
- Most common example of a static member is **main()**
- They can call only other **static** methods
- They cannot refer to **this** & **super** in any ways

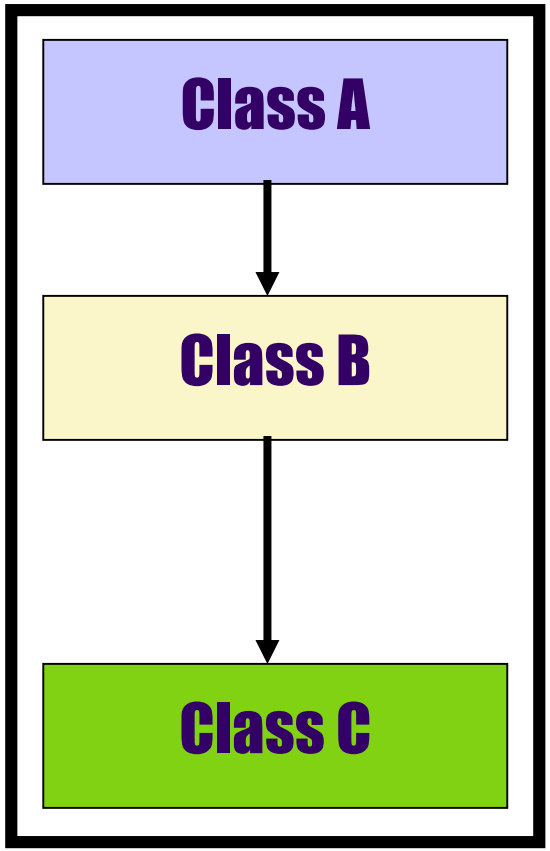
Inheritance Basics



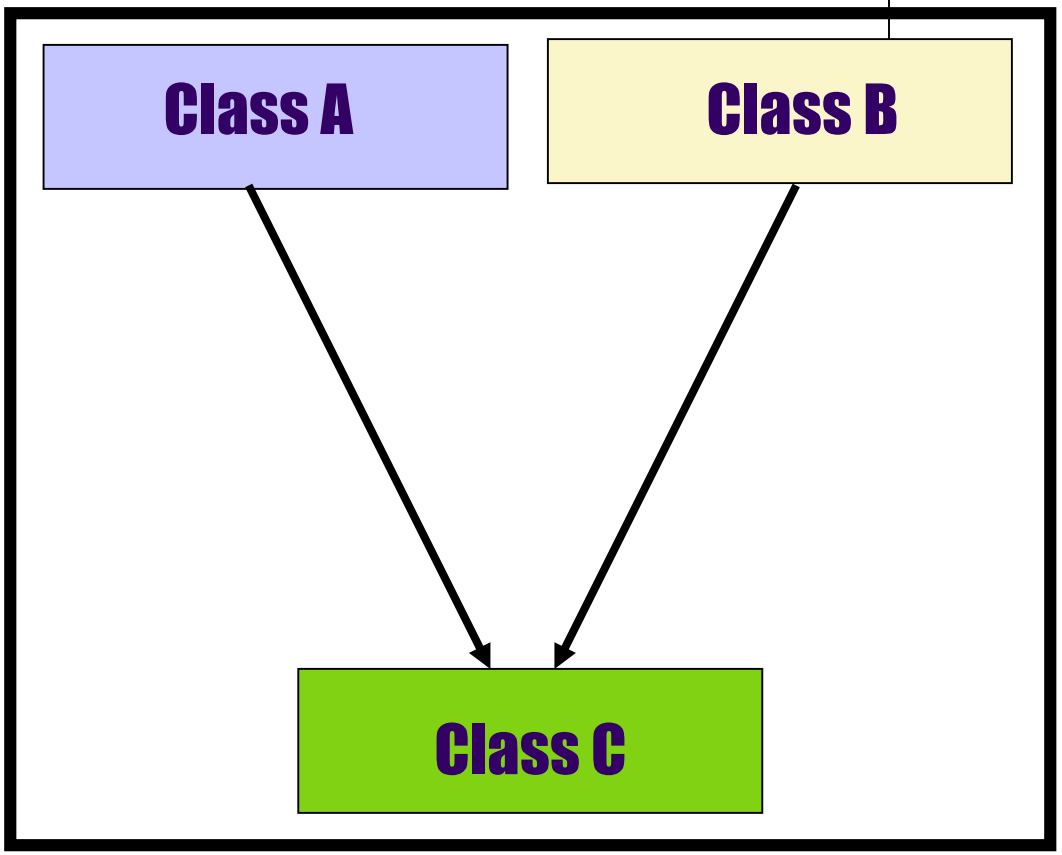
- Incorporate the definition of one class into another by using the **extends** keyword.
- A super class is completely independent, stand-alone class.
- Further, a subclass can be a superclass for another subclass
- You can specify one superclass for any subclass that you create
- Member Access & Inheritance

```
class subclass-name extends superclass-name{
        //body of class
}
```

Inheritance in Java

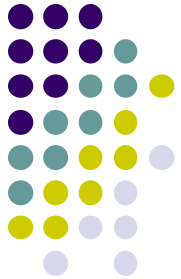


Allowed in Java



Not Allowed in Java

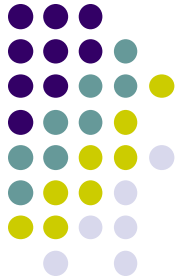
Method Overriding



A method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to *override* the method in the superclass.

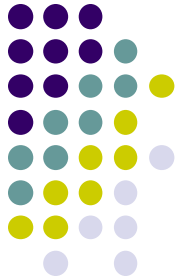
- Dynamic Method Dispatch
  - ✓ Call to an overridden function is resolved at run time, rather than compile time, this is how JAVA implements runtime **polymorphism**

Overloading vs. Overriding



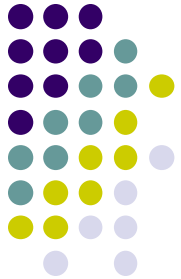
- Don't confuse the concepts of overloading and overriding
- Overloading deals with multiple methods in the same class with the same name but different signatures
- Overriding deals with two methods, one in a parent class and one in a child class, that have the same signature
- Overloading lets you define a similar operation in different ways for different data
- Overriding lets you define a similar operation in different ways for different object types

Abstract Class



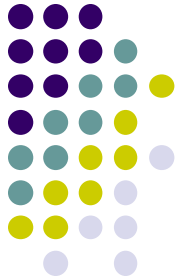
- To declare a class abstract, simply use **abstract** keyword.
- Any **class** that contains one or more **abstract** methods must also be declared **abstract**.
- Cannot be instantiated
- Abstract constructors and abstract static methods cannot be declared in a abstract class
- Any subclass of an abstract class must either implement all the abstract methods in the super class, or it should be declared abstract

Final Modifier



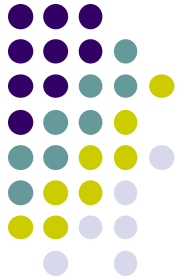
- **“final”** modifier has a meaning based on its usage
  - ✓ For variable: the value is constant
  - ✓ For methods: such methods cannot be overridden
  - ✓ For class: such class cannot be sub-classed

Binding



- Java resolves calls to methods dynamically, at runtime. This is called as *late binding*. However, since final methods cannot be overridden, a call to one can be resolved at compile time. This is called *early binding*

Prevent Inheritance



```
final class A{
    // . . . .
}
// The following class is illegal
Class B extends A{ // ERROR! Can't be subclass of A
    //. . . .
}
```

Package

Why Use Packages ?

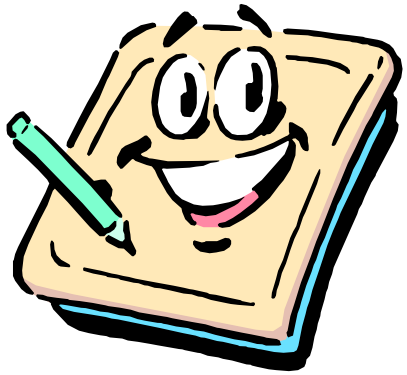
Just think of Writing the code from the scratch, each time you create an application

You'll end up spending your precious time and energy and finally land up with a Huge code accumulated before you.

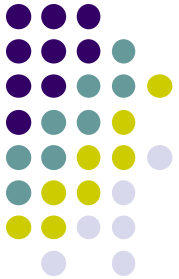
Reusing The Existing Code

- ✓ Reusability of code is one of the most important requirements in the software industry.
- ✓ Reusability saves time, effort and also ensures consistency.

A class once developed can be reused by any number of programs wishing to incorporate the class in that particular program.



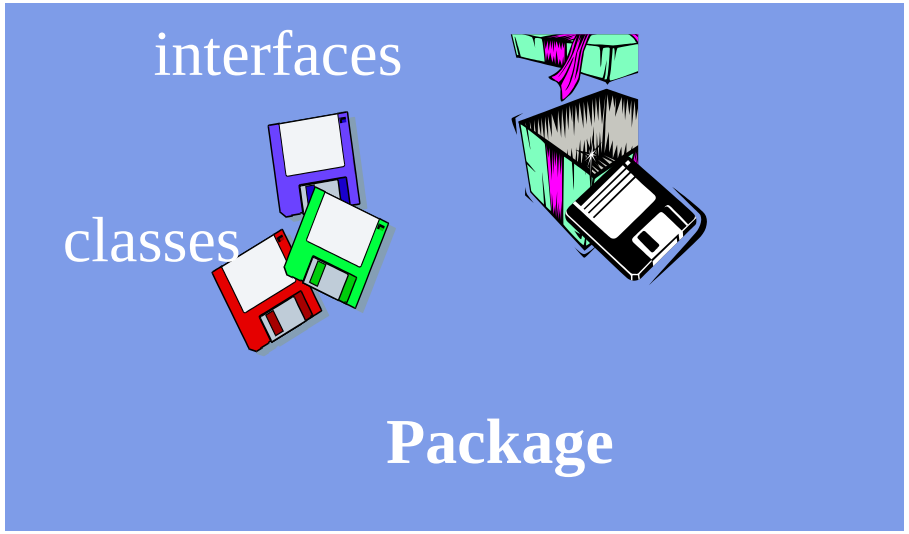
Reusability



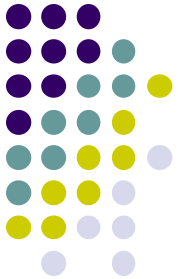
- Reusability saves time, effort and also ensures **consistency**
- Need not write code from the **scratch**, for each application
- Reusability of code is one of the most important requirements in the software industry today.
- A class once developed can be reused by any number of programs
- In Java, the codes which can be reused by other programs is put into a "**Package**".

Package Advantages

- In Java, the codes which can be reused by other programs is put into a "**Package**".
- A Package is a collection of **classes**, **interfaces** and/or other **packages**.
- Packages are essentially a means of organizing classes together as **groups**.

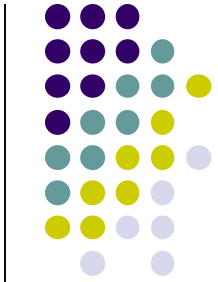


Package Advantages…..



- Packages allow you to organize your classes into smaller units (such as folders) and make it easy to locate and use the appropriate class file.
- It helps to avoid naming conflicts
- Logical Grouping of classes

Compiling the Package



- **javac -d c:\ temp Calculate.java**

  When the above command is executed on the command prompt, the compiler creates a folder called “mypackage” in the temp directory and stores the “Calculate.class” into this folder

- **javac -d . Calculate.java**

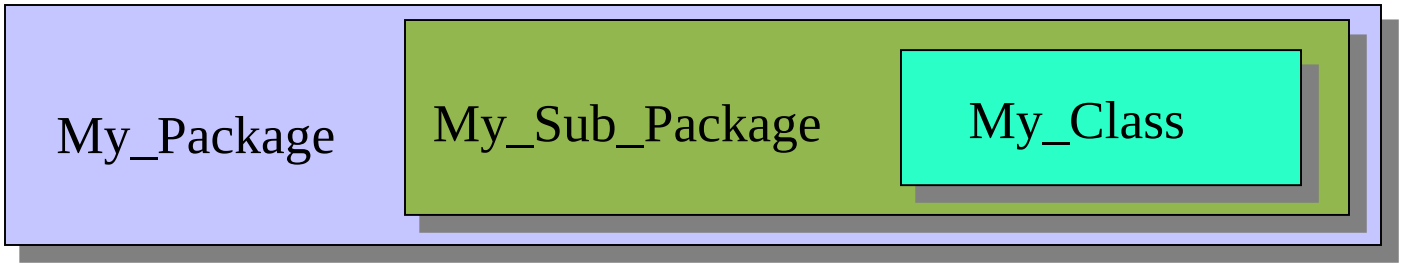
  When the above command is executed on the command prompt, the compiler creates a folder called “mypackage” in the current directory and stores the “Calculate.class” into this folder

***Note**: the statement for creating a package must be written before any other import statements*

Importing a Package

```
import packagename.classname;
```

- Suppose the class is located within a sub package like this:

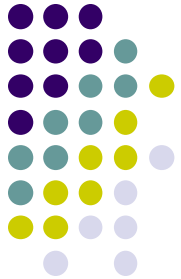



```
import MyPackage.MySubPackage.MyClass
Import java.awt.*
Import java.awt.event.*
```

Important Packages in Java

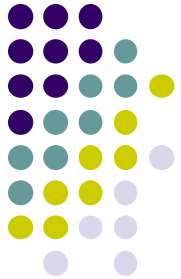
java.lang	You don't need to explicitly import this package. It is always imported for you.
java.io	This package consists of classes that help you for all the Input and Output operations.
java.applet	This package consists of classes that you need, to execute an applet in the browser or an appletviewer.
java.awt	This package is useful to create GUI applications.
java.util	This package provides a variety of classes and interfaces for creating lists, calendar, date, etc.
java.net	This package provides classes and interfaces for TCP/IP network programming.

Interfaces



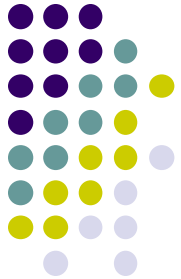
- Multiple inheritance in java is achieved through interface
- An **interface** in the Java programming language is an abstract type
- Classes should implement **`interface`**
- May only contain method signatures and constant declarations
- They cannot be directly instantiated
- The class must implement all of the methods described in the interface
- An interface may extend, any number of interfaces; however an interface may not implement an interface.

Defining an Interface



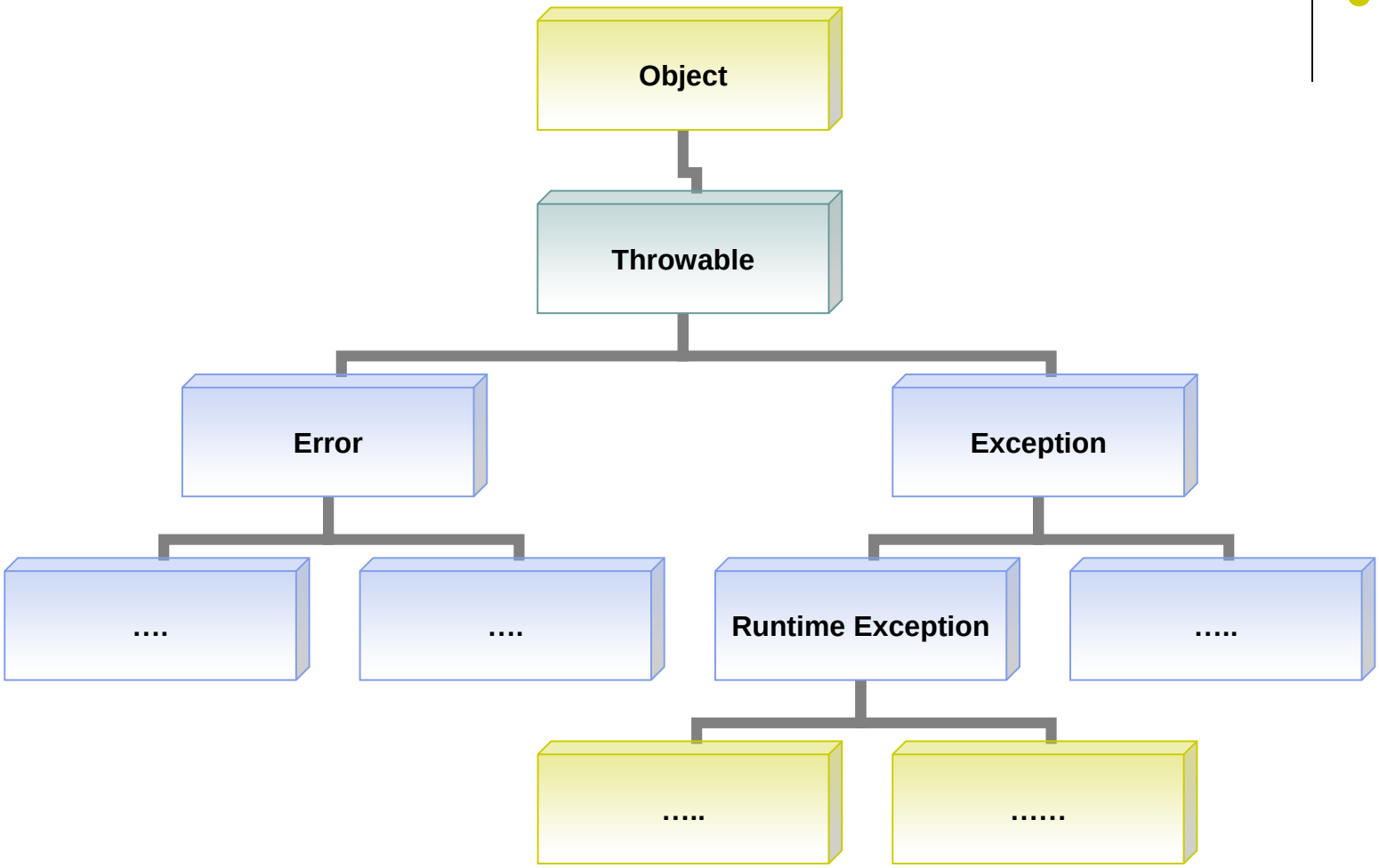
```
access interface name [extends other interfaces] {
        return-type method-name1(parameter-list);
        return-type method-name2(parameter-list);
        type final-varname1=value;
        type final-varname2=value;
        //….
}
```

Exception Handling

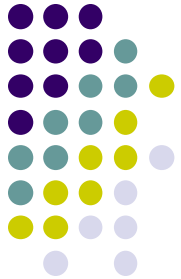


- Basic concepts of Exception Handling
- Errors and Exceptions
- Try – Catch – Finally block
- Throw & Throws
- Some Built-in exceptions
- User defined exceptions

The Hierarchy

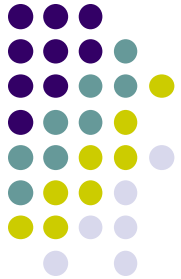


Uncaught Exceptions



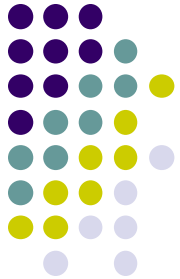
- `int i=5/0;`
- What happens when the above code is executed ?
- Exception is thrown (an object is thrown)
- How to recover from this ? (handle it !)

Exceptions



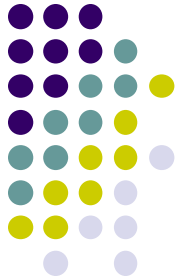
- An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.
- The exception object contains information about the exception, including its type and the state of the program when the error occurred.
- Helpful in separating the execution code from the error handler
- The Java programming language provides a mechanism known as exceptions to help programs report and handle errors.
- When an error occurs, the program throws an exception.

Exceptions,…….



- The Java programming language provides a mechanism known as exceptions to help programs report and handle errors.
- When an error occurs, the program throws an exception.
- Normal flow of the program is interrupted
- The runtime attempts to find an exception handler
- Three statements play a part in handling exceptions: try – catch – finally
- Java enforces exceptions to be handled by the programmer. If not, compile error occurs.

Exceptions & Errors



- Exceptions are situations within the control of an application, that it should try to handle
- Errors indicate serious problems and abnormal conditions that most applications should not try to handle

try – catch - finally

```
try
        {
        ……
        ……
        …… }
catch (exceptionType name)
        {
        ……
        ……
        …… }
finally
        {
        ……
        ……
        …… }
```

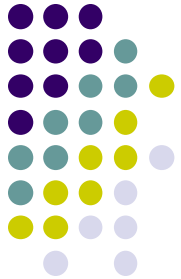
Multiple Clauses

Multiple catch Clauses

Nested statements

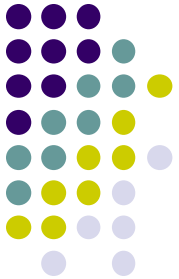
Nested try Statements

Finally Block



- The finally statement is associated with a try statement and identifies a block of statements that are executed regardless of whether or not an error occurs within the try block.
- Defines the code that is executed always
- In the normal execution it is executed after the try block
- When an exception, it is executed after the handler or before propagation as the case may be

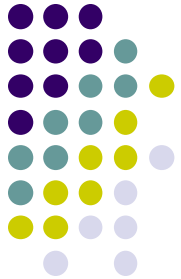
Throwing Exceptions



- Use the throw clause to throw an exception
- Exceptions in Java are compulsorily of type Throwable
- If you attempt to throw an object that is not throwable, the compiler refuses to compile your program
- Can also be used to re-throw an exception

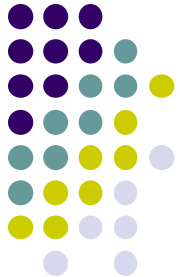
```
public void read() throws IOException
        {
            // Some code that cause IO Exception
            throw new IOException();
        }
```

throws



- The throws keyword is used along with the declaration of a method that can throw an exception.
- This makes it mandatory for anyone calling the method to have it in try block else the compiler will give an error.

User Defined Exceptions



- We can create our own exception objects, which can be thrown using the throw keyword
- Create a class which extends the Exception class [this is our user-defined exception class, that we want to throw]
- Create an instance of the user-defined exception class.
- Use the throw keyword to throw the instance created !
- Catch it accordingly.

Thank You!