Core Java

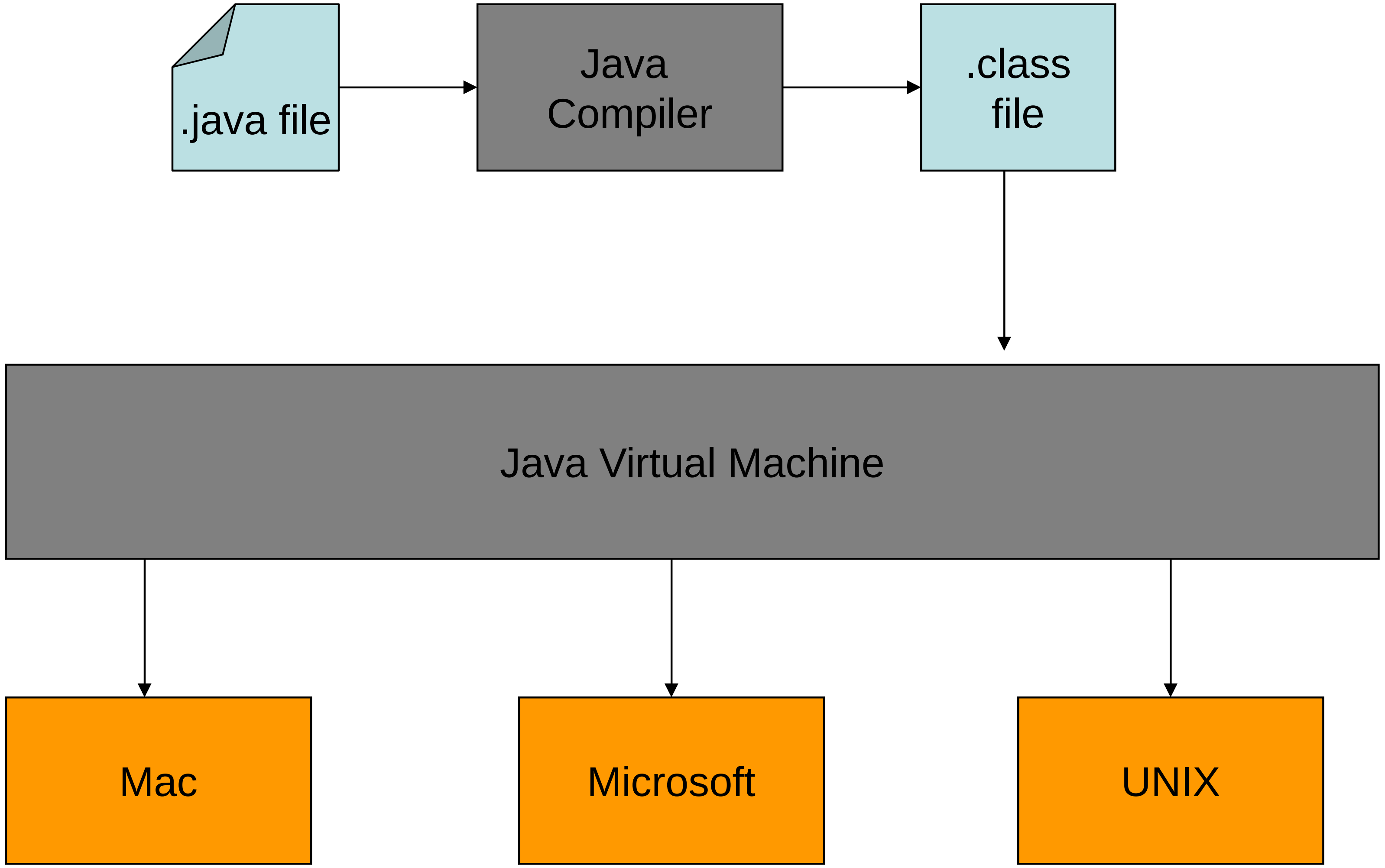
Agenda

- Introduction
- Access Modifiers
- Operators
- Flow Control
- Arrays and Strings
- OOPS Explored
- Exceptions
- Garbage Collection
- Collections
- Threads
- Demo

Introduction – What is Java

- Programming language
  - Another programming language using which we can develop applets, standalone applications, web applications and enterprise applications.
- Platform Independent
  - A Java program written and compiled on one machine can be executed on any other machine (irrespective of the operating system)
- Object Oriented
  - Complies to object oriented programming concepts. Your program is not object oriented unless you code that way
- Compiled and Interpreted
  - The .java file is compiled to a .class file & the .class file is interpreted to machine code

Introduction - Java Virtual Machine



Object Oriented Concepts in Java

Introduction - Object Oriented Concepts

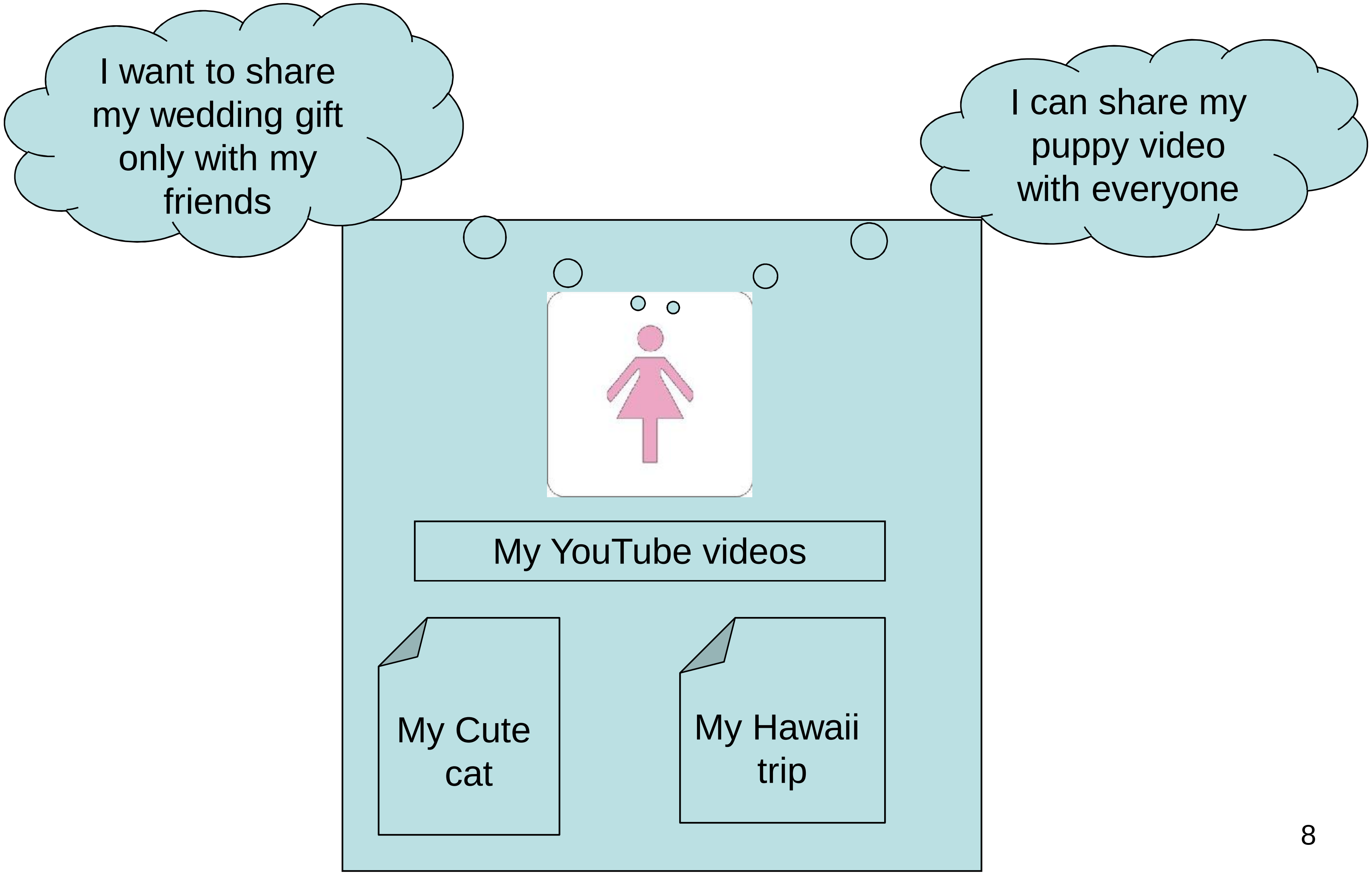
- Class
  - A blueprint that defines the attributes and methods. The class is a logical group of similar entities
- Object
  - An instance of a Class
- Abstraction
  - Hide certain details and show only essential details
- Encapsulation
  - Encapsulation is the technique used to implement abstraction in object oriented programming.
  - Encapsulation is used for access restriction to a class members and methods.
  - Binding data and methods together

Core OOPS concepts:

- Polymorphism
  - One name having many forms
  - Polymorphism is the concept where an object behaves differently in different situations
  - polymorphism is achieved by method overloading or method overriding

- Inheritance
  - Inherit the features of the superclass
  - Inheritance is the mechanism of code reuse.
  - The object that is getting inherited is called superclass and the object that inherits the superclass is called subclass.
  - We use extends keyword in java to implement inheritance

OOPS Explored - Encapsulation


I want to share my wedding gift only with my friends
I can share my puppy video with everyone
My YouTube videos
My Cute cat
My Hawaii trip

OOPS Explored - Encapsulation

- **Encapsulation**
  Binding data and methods together

```
public class Employee
{
    private String empName;
    private int salary;

    public String getSalary(String role)
    {
        if(“Manager”.equals(role))  {
            return salary;
        }
    }

    public String setSalary(String role, int newSal)
    {
        if (“Admin”.equals(role))  {
            salary = newSal;
        }
    }
}
```

OOPS Explored - Polymorphism

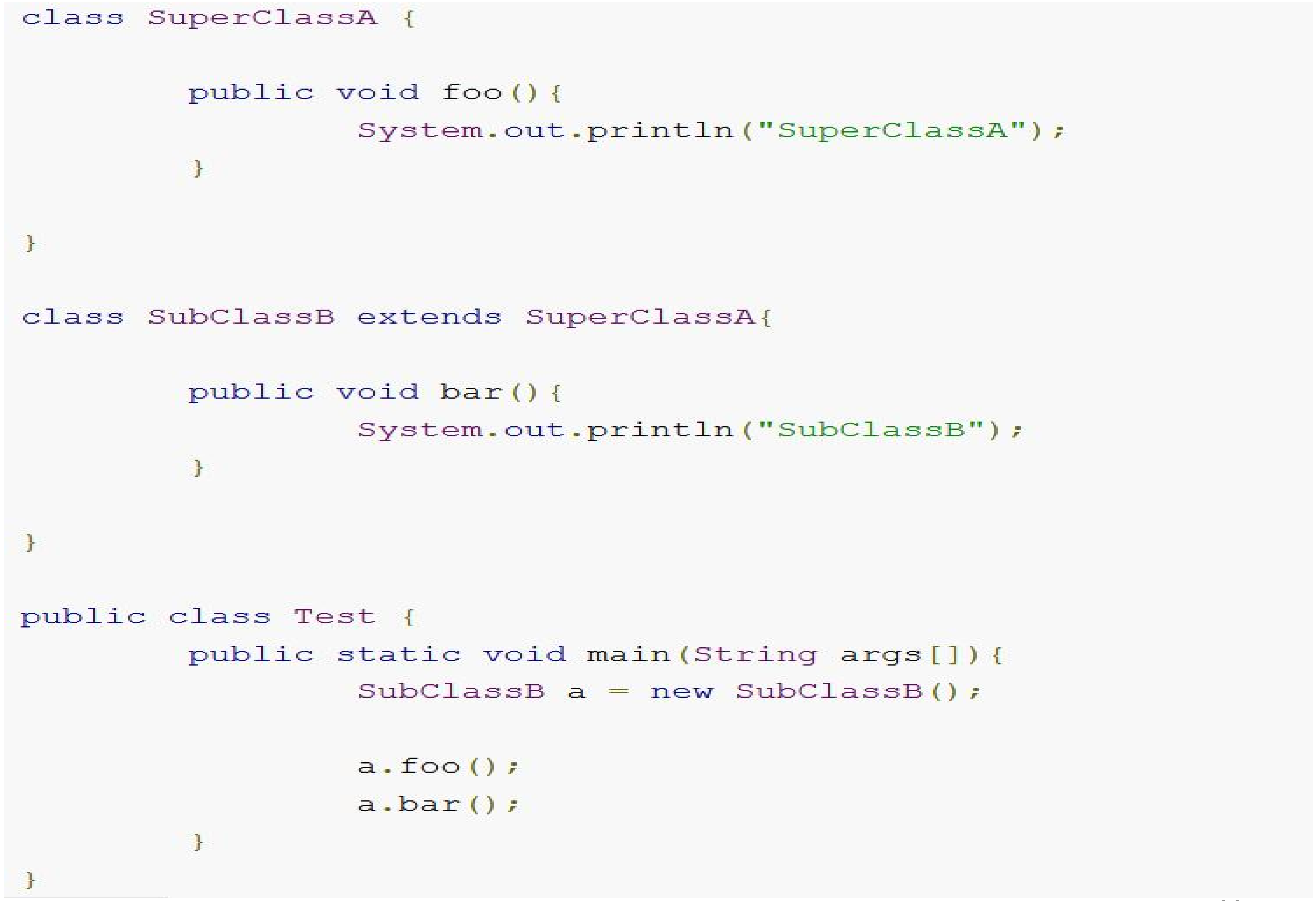
```
public class Circle {

        public void draw(){
                System.out.println("Drwaing circle with default color Black and
diameter 1 cm.");
        }

        public void draw(int diameter){
                System.out.println("Drwaing circle with default color Black and
diameter"+diameter+" cm.");
        }

        public void draw(int diameter, String color){
                System.out.println("Drwaing circle with color"+color+" and
diameter"+diameter+" cm.");
        }
}
```

OOPS Explored - Inheritance



```
class SuperClassA {

        public void foo(){
                System.out.println("SuperClassA");
        }


}

class SubClassB extends SuperClassA{

        public void bar(){
                System.out.println("SubClassB");
        }


}

public class Test {
        public static void main(String args[]){
                SubClassB a = new SubClassB();

                a.foo();
                a.bar();
        }
}
```

My First Program Version 1

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println(“Hello World”);
  }
}
```

Compile the program: javac HelloWorld.java
Execute the program: java HelloWorld
Output: Hello World

My First Program Version 2

```
package com.smitaExample.test;

public class HelloWorld
{
  public static void main(String[] args)
  {
    HelloWorld hw = new HelloWorld();
    hw.display();
  }

  public void display()
  {
    System.out.println("Hello World");
  }
}
```

Compile the program: javac HelloWorld.java
Execute the program: java HelloWorld
Output: Hello World

Introduction – Java Keywords

abstract	boolean	break	byte	case	catch	char
class	const	continue	default	do	double	else
extends	final	finally	float	for	goto	if
implements	import	instanceof	int	interface	long	native
new	package	private	protected	public	return	short
static	strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while	assert

Introduction - Data Types

Data type	Bytes
byte	1
short	2
int	4
long	8
float	4
double	8
char	2
boolean	True/false-

Java Modifiers

Modifier	Class	Class Variables	Methods	Method Variables
public	✓	✓	✓	
private		✓	✓	
protected		✓	✓	
default	✓	✓	✓	
final	✓	✓	✓	✓
abstract	✓		✓	

Modifiers – Class

- public
  - Class can be accessed from any other class present in any package
- default
  - Class can be accessed only from within the same package. Classes outside the package in which the class is defined cannot access this class
- final
  - This class cannot be sub-classed, one cannot extend this class
- abstract
  - Class cannot be instantiated, need to sub-classs/extend.

Modifiers – Class Attributes

- public
  - Attribute can be accessed from any other class present in any package
- private
  - Attribute can be accessed from only within the class
- protected
  - Attribute can be accessed from all classes in the same package and sub-classes.
- default
  - Attribute can be accessed only from within the same package.
- final
  - This value of the attribute cannot be changed, can assign only 1 value
- static
  - Only one value of the attribute per class

Modifiers – Methods

- public
  - Method can be accessed from any other class present in any package
- private
  - Method can be accessed from only within the class
- protected
  - Method can be accessed from all classes in the same package and sub-classes.
- default
  - Method can be accessed only from within the same package.
- final
  - The method cannot be overridden
- abstract
  - Only provides the method declaration
- static
  - Can access only static members.

Static - [1/4]

Member data –

- Same data is used for all the instances (objects) of some Class.

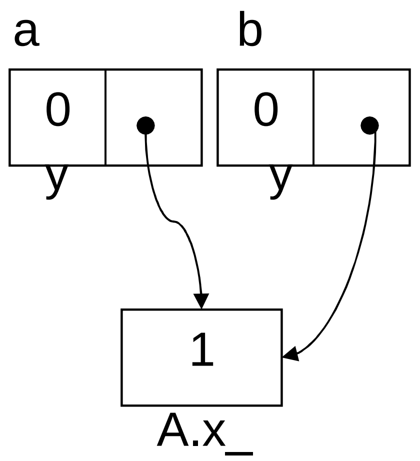
```
Class A {
   public int y = 0;
   public static int x_ = 1;
};

A a = new A();
A b = new A();
System.out.println(b.x_);
a.x_ = 5;
System.out.println(b.x_);
A.x_ = 10;
System.out.println(b.x_);
```

Assignment performed on the first access to the Class.
Only one instance of 'x' exists in memory

Output:

```
1
5
10
```



Static - [2/4]

Member function

- Static member function can access only static members
- Static member function can be called without an instance.

```
Class TeaPot {
        private static int numOfTP = 0;
        private Color myColor_;
        public TeaPot(Color c) {
                myColor_ = c;
                numOfTP++;
        }
        public static int howManyTeaPots()
                { return numOfTP; }
        // error :
        public static Color getColor()
                { return myColor_; }
}
```

Usage:

```
TeaPot tp1 = new TeaPot(Color.RED);

TeaPot tp2 = new TeaPot(Color.GREEN);

System.out.println(“We have “ +
        TeaPot.howManyTeaPots()+ “Tea
Pots”);
```

Operators - Types

- Definition:

  An operator performs a particular operation on the operands it is applied on

- Types of operators
  - Assignment Operators
  - Arithmetic Operators
  - Unary Operators
  - Equality Operators
  - Relational Operators
  - Conditional Operators

Operators – Assignment Operators/Arithmetic Operators

- Assignment Operator

Operator	Description	Example

=	Assignment	int i = 10;
		int j = i;

- Arithmetic Operators

Operator	Description	Example
+	Addition	int i = 8 + 9; byte b = (byte) 5+4;
-	Subtraction	int i = 9 – 4;
*	Multiplication	int i = 8 * 6;
/	Division	int i = 10 / 2;
%	Remainder	int i = 10 % 3;

Operators – Unary Operators/Equality Operators

- Unary Operators

Operator	Description	Example
+	Unary plus	int i = +1;
-	Unary minus	int i = -1;
++	Increment	int j = i++;
--	Decrement	int j = i--;
!	Logical Not	boolean j = !true;

- Equality Operators

Operator	Description	Example
==	Equality	If (i==1)
!=	Non equality	If (i != 4)

Operators – Relational Operators/Conditional Operators

- Relational Operators

Operator	Description	Example
>	Greater than	if (x > 4)
<	Less than	if (x < 4)
>=	Greater than or equal to	if (x >= 4)
<=	Less than or equal to	if (x <= 4)

- Conditional Operators

Operator	Description	Example
&&	Conditional and	If (a == 4 && b == 5)
||	Conditional or	If (a == 4 || b == 5)

Flow Control – if-else if-else

- **if-else**

Syntax	Example
if (<condition-1>) {	int a = 10;
// logic for true condition-1 goes here	if (a < 10) {
} else if (<condition-2>) {	System.out.println(“Less than 10”);
// logic for true condition-2 goes here	} else if (a > 10) {
} else {	System.out.pritln(“Greater than 10”);
// if no condition is met, control comes here	} else {
}	System.out.println(“Equal to 10”);
	}
	Result: Equal to 10s

Flow Control – switch

- **switch**

Syntax	Example

```
switch (<value>)  {
  case <a>:
    // stmt-1
    break;
  case <b>:
    //stmt-2
    break;
  default:
    //stmt-3
```
	```
	int a = 10;
	switch (a)  {
	  case 1:
	    System.out.println(“1”);
	    break;
	  case 10:
	    System.out.println(“10”);
	    break;
	  default:
	    System.out.println(“None”);

	Result: 10
	```

Flow Control – do-while / while

- **do-while**

Syntax	Example
do {	int i = 0;
// stmt-1	do {
} while (<condition>);	System.out.println(“In do”); i++;
	} while (i < 10);
	Result: Prints “In do” 11 times

- **while**

Syntax	Example
while (<condition>) {	int i = 0;
//stmt	while (i < 10) {
}	System.out.println(“In while”); i++;
	}
	Result: “In while” 10 times

Flow Control – for loop

- **for**

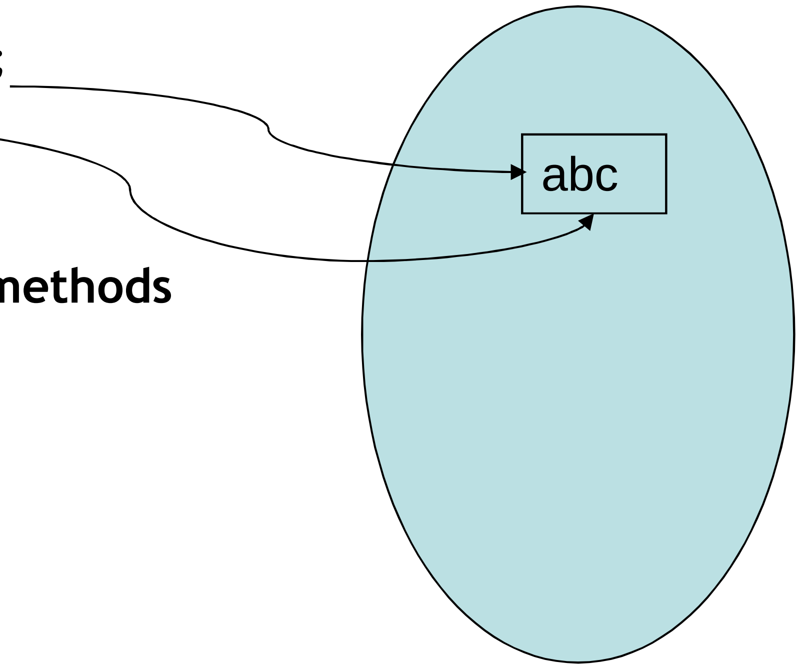
Syntax	Example

for (initialize; condition; expression)
{
   // stmt
}
	for (int i = 0; i < 10; i++)
	{
	  System.out.println(“In for”);
	}
	
	Result: Prints “In do” 10 times

Strings

- **Creating String Objects**

String myStr1 = new String(“abc”);
String myStr2 = “abc”;



String Constant Pool

- **Most frequently used String methods**

  - charAt (int index)
  - compareTo (String str2)
  - concat (String str2)
  - equals (String str2)
  - indexOf (int ch)
  - length()
  - replace (char oldChar, char newChar)
  - substring (int beginIndex, int endIndex)

Constructors

- Creates instances for Classes
- Same name as Class name
- Can have any access modifier
- First statement should be a call to this() or super()

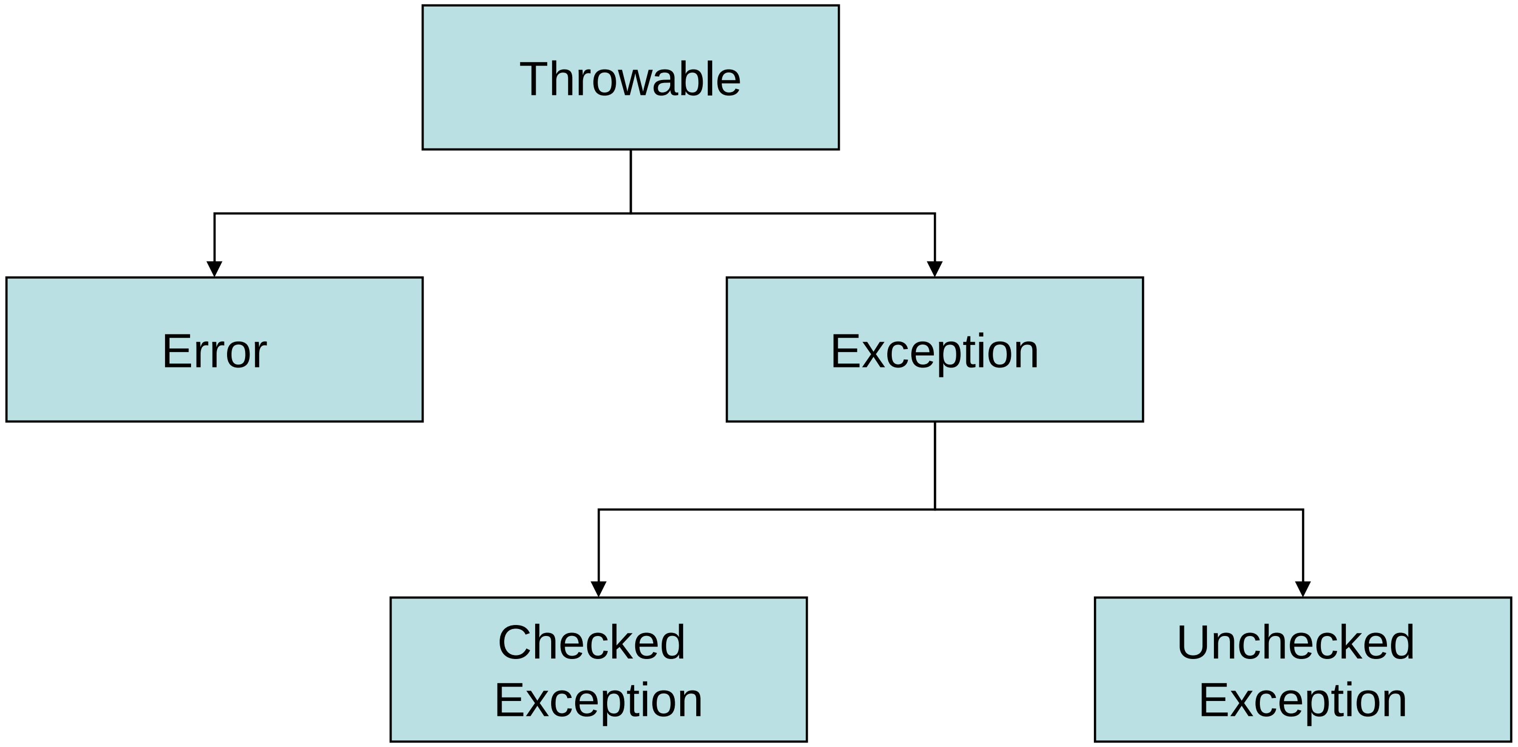
```
Employee emp = new Employee()
```

```
public class Employee  {
   public int empid;
   public String name;

   public Employee(int empid) {
     this.empid = empid;
   }

   public Employee(String name, int empid) {
     this.name = name;
     this.empid = empid;
   }
}
```

Exceptions – Exception Hierarchy



Exceptions - Handling exceptions

- **What do you do when an Exception occurs?**
  - Handle the exception using try/catch/finally
  - Throw the Exception back to the calling method.

- **Try/catch/finally**

```
public class MyClass  {
  public void exceptionMethod()  {
      try  {
         // exception-block
      } catch (Exception ex)  {
         // handle exception
      } finally  {
         //clean-up
      }
  }
}
```

Exceptions – Handling exceptions

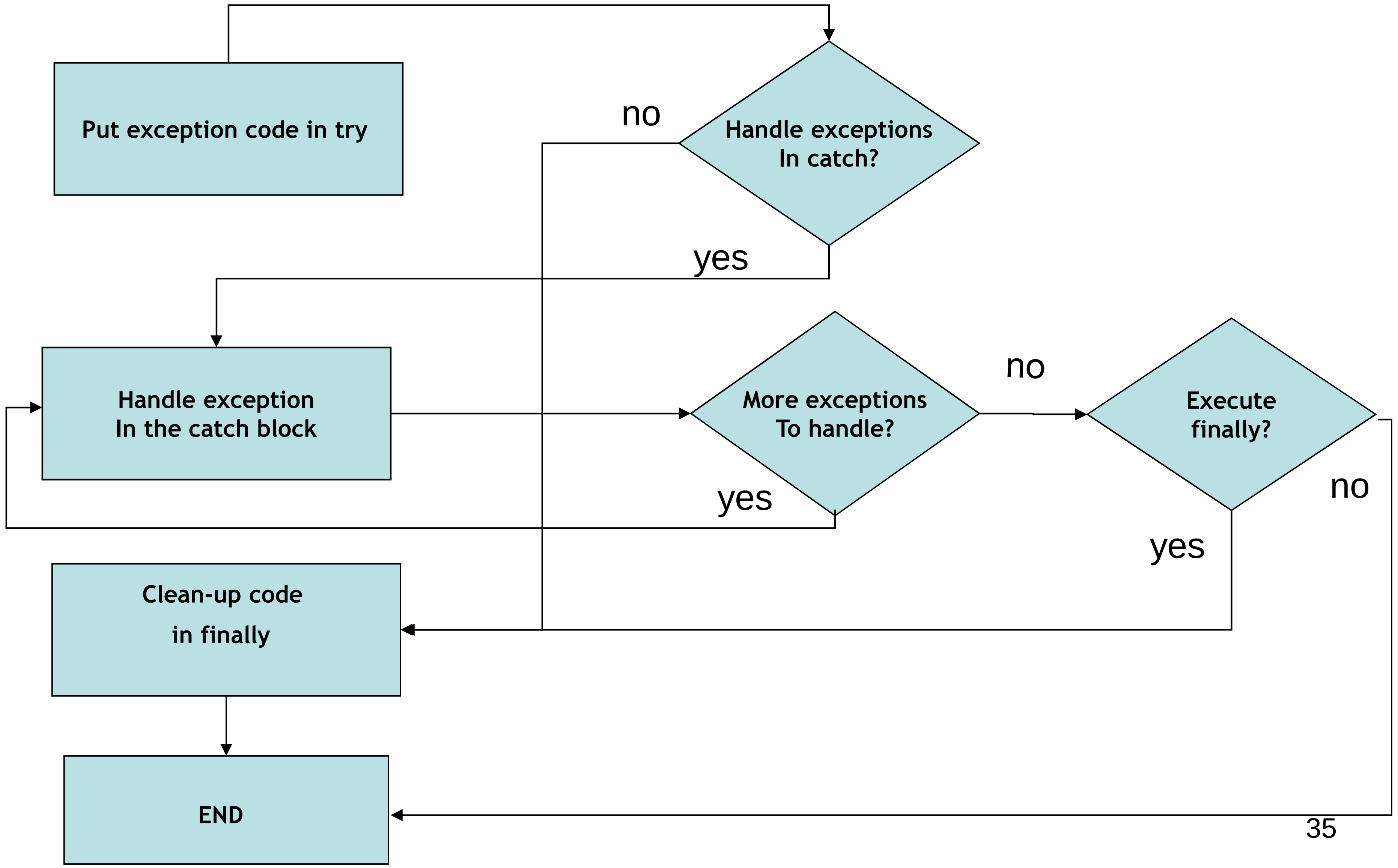
- **Try/catch/finally - 2**

```
public class MyClass  {
  public void exceptionMethod()  {
      try  {
         // exception-block
      } catch (FileNotFoundException ex)  {
         // handle exception
       } catch (Exception ex)  {
         // handle exception
      } finally  {  //clean-up  }
  }
}
```

- **Using throws**

```
public class MyClass  {
  public void exceptionMethod() throws Exception {
     // exception-block
  }
}
```

Exceptions – try-catch-finally flow



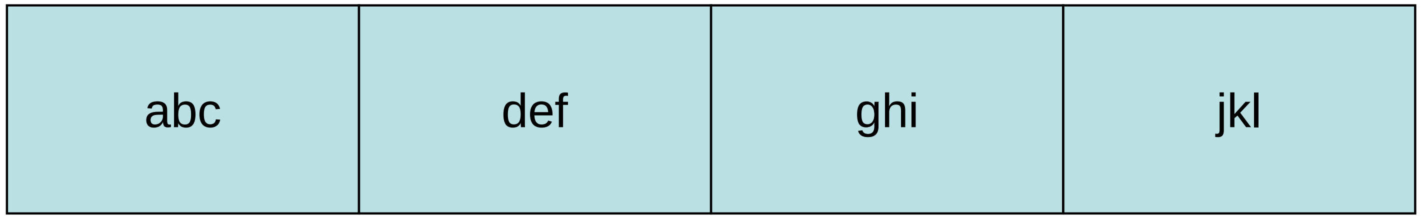
Put exception code in try
Handle exceptions In catch?
no
yes
Handle exception In the catch block
More exceptions To handle?
no
yes
Execute finally?
no
yes
Clean-up code in finally
END

Collections - Introduction

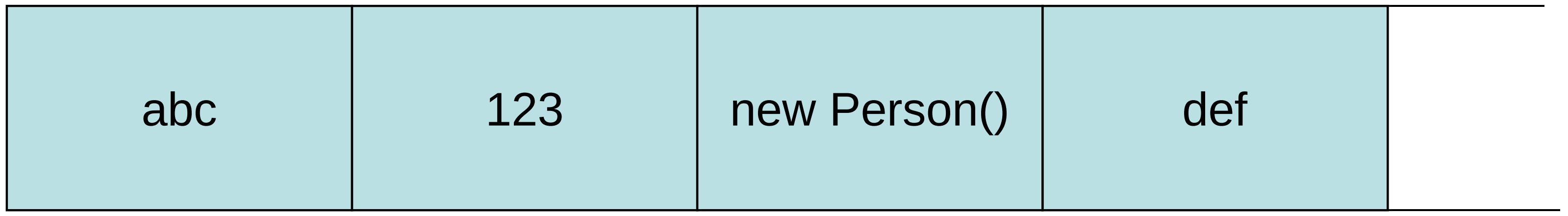
```
String student1 = “a”;
String student2 = “b”;
String student3 = “c”;
String student4 = “d”;
String student5 = “e”;
String student6 = “f”;
```

- **Difficult to maintain multiple items of same type as different variables**
- **Data-manipulation issues**
- **Unnecessary code**

Collections – Arrays v/s Collections



Arrays



Collections

Collections – Collection types

- **Flavors of collections:**

  Lists - Lists of things (classes that implement List)

  Sets - Unique things (classes that implement Set)