# Einführung in die Programmierung

## Java Quiz

ETH zürich

# Grouping Exceptions

```
public class TryCatch {
    public static void main(String[] args) {
        System.out.println("main()");
        new TryCatch().m();
    }

    public void m() {
        try {
            System.out.println("m(): try");
            int i = 1 / 0;
        } catch (NullPointerException e) {
            System.out.println("m(): catch NullPE");
        } catch (ArithmeticException e) { // Div-by-0
            System.out.println("m(): catch DivBy0");
        } catch (Exception e) {
            System.out.println("m(): catch Exc");
        }
    }
}
```

Output:

# Grouping Exceptions

```
public class TryCatch {
    public static void main(String[] args) {
        System.out.println("main()");
        new TryCatch().m();
    }

    public void m() {
        try {
            System.out.println("m(): try");
            int i = 1 / 0;
        } catch (NullPointerException e) {
            System.out.println("m(): catch NullPE");
        } catch (ArithmeticException e) { // Div-by-0
            System.out.println("m(): catch DivBy0");
        } catch (Exception e) {
            System.out.println("m(): catch Exc");
        }
    }
}
```

## Output:

```
main()
m():  try
m():  catch DivBy0
```

# Grouping Exceptions

```
public class TryCatch {
    public static void main(String[] args) {
        System.out.println("main()");
        new TryCatch().m();
    }

    public void m() {
        try {
            System.out.println("m(): try");
            int i = 1 / 0;
        } catch (NullPointerException | ArithmeticException e) {
            System.out.println("m(): catch known Err");
        } catch (Exception e) {
            System.out.println("m(): catch Exc");
        }
    }
}
```

Output:

# Grouping Exceptions

```java
public class TryCatch {
    public static void main(String[] args) {
        System.out.println("main()");
        new TryCatch().m();
    }

    public void m() {
        try {
            System.out.println("m(): try");
            int i = 1 / 0;
        } catch (NullPointerException | ArithmeticException e) {
            System.out.println("m(): catch known Err");
        } catch (Exception e) {
            System.out.println("m(): catch Exc");
        }
    }
}
```

## Output:

```
main()
m():  try
m():  catch known Err
```

# Grouping Exceptions

```
public class TryCatch {
    public static void main(String[] args) {
        System.out.println("main()");
        new TryCatch().m();
    }

    public void m() {
        try {
            System.out.println("m(): try");
            int i = 1 / 0;
        } catch (RuntimeException | ArithmeticException e) {
            System.out.println("m(): catch known Err");
        } catch (Exception e) {
            System.out.println("m(): catch Exc");
        }
    }
}
```

Output:

# Grouping Exceptions

```
public class TryCatch {
    public static void main(String[] args) {
        System.out.println("main()");
        new TryCatch().m();
    }

    public void m() {
        try {
            System.out.println("m(): try");
            int i = 1 / 0;
        } catch (RuntimeException | ArithmeticException e) {
            System.out.println("m(): catch known Err");
        } catch (Exception e) {
            System.out.println("m(): catch Exc");
        }
    }
}
```

## Output:

Unresolved compilation problem: The exception `ArithmeticException` is already caught by the alternative `RuntimeException`

# Something Peculiar

```
public class C {
    static C obj;
    C() {
        C.obj = this;
        throw new RuntimeException();
    }

    public static void main(String[] args) {
        C o = null;
        try {
            o = new C();
        } catch (RuntimeException e) {
            /* ignore */
        }
        System.out.println(C.obj);
        System.out.println(o);
    }
}
```

Output:

# Something Peculiar

```
public class C {
    static C obj;
    C() {
        C.obj = this;
        throw new RuntimeException();
    }

    public static void main(String[] args) {
        C o = null;
        try {
            o = new C();
        } catch (RuntimeException e) {
            /* ignore */
        }
        System.out.println(C.obj);
        System.out.println(o);
    }
}
```

## Output:

ClassName@hashCode
null

**Reason:** Instantiation of object failed in the constructor.
1 2

[1] https://stackoverflow.com/questions/36376339/how-does-object-tostring-get-the-memory-address-and-how-can-i-mimic-it

[2] https://stackoverflow.com/questions/5909818/java-exception-thrown-in-constructor-can-my-object-still-be-created

# Polymorphism

What access modifiers can we give one method while keeping the other method untouched?

```
class Base {
    void myMethod() {}
}

class Derived extends Base {
    void myMethod() {}
}
```

## Output

# Polymorphism

What access modifiers can we give one method while keeping the other method untouched?

```
1 class Base {
2     void myMethod() {}
3 }
4
5 class Derived extends Base {
6     void myMethod() {}
7 }
```

## Output

Line 3: private/*no modifier*
Line 8: *no modifier*/protected/public

# Polymorphism

```
class Base {
    public void show() {
       System.out.println("Base::show() called");
    }
}

class Derived extends Base {
    public void show() {
       System.out.println("Derived::show() called");
    }
}

public class Main {
    public static void main(String[] args) {
        Base b = new Derived();;
        b.show();
    }
}
```

## Output

# Polymorphism

```
class Base {
    public void show() {
       System.out.println("Base::show() called");
    }
}

class Derived extends Base {
    public void show() {
       System.out.println("Derived::show() called");
    }
}

public class Main {
    public static void main(String[] args) {
        Base b = new Derived();;
        b.show();
    }
}
```

## Output

Derived::show() called

# Polymorphism

```
class Base {
    final public void show() {
       System.out.println("Base::show() called");
    }
}

class Derived extends Base {
    public void show() {
       System.out.println("Derived::show() called");
    }
}

public class Main {
    public static void main(String[] args) {
        Base b = new Derived();;
        b.show();
    }
}
```

## Output

# Polymorphism

```
class Base {
    final public void show() {
       System.out.println("Base::show() called");
    }
}

class Derived extends Base {
    public void show() {
       System.out.println("Derived::show() called");
    }
}

public class Main {
    public static void main(String[] args) {
        Base b = new Derived();;
        b.show();
    }
}
```

## Output

error: show() in Derived
cannot override show() in Base
overridden method is final

# Polymorphism

```
class Base {
    public static void show() {
       System.out.println("Base::show() called");
    }
}

class Derived extends Base {
    public static void show() {
       System.out.println("Derived::show() called");
    }
}

public class Main {
    public static void main(String[] args) {
        Base b = new Derived();;
        b.show();
    }
}
```

Output

# Polymorphism

```
class Base {
    public static void show() {
        System.out.println("Base::show() called");
    }
}

class Derived extends Base {
    public static void show() {
        System.out.println("Derived::show() called");
    }
}

public class Main {
    public static void main(String[] args) {
        Base b = new Derived();;
        b.show();
    }
}
```

## Output

Base::show() called

No runtime polymorphism in Java for static methods as they are class specific, not instance specific. In this instance the subclass' method hides the superclass' method.
[3]

[3] https://docs.oracle.com/javase/tutorial/java/IandI/override.html

# Polymorphism

Which of these are valid?

1. Private methods are final.
2. Protected members are accessible within a package and inherited classes outside the package.
3. Protected methods are final.
4. We cannot override private methods.

Output

# Polymorphism

Which of these are valid?

1. Private methods are final.
2. Protected members are accessible within a package and inherited classes outside the package.
3. Protected methods are final.
4. We cannot override private methods.

## Output

1, 2 and 4

# Polymorphism

```
class Base {
    public void Print() {
        System.out.println("Base");
    }
}

class Derived extends Base {
    public void Print() {
        System.out.println("Derived");
    }
}

public class Main {
    public static void DoPrint(Base o) {
        o.Print();
    }

    public static void main(String[] args) {
        Base x = new Base();
        Base y = new Derived();
        Derived z = new Derived();
        DoPrint(x);
        DoPrint(y);
        DoPrint(z);
    }
}
```

Output

# Polymorphism

```
class Base {
    public void Print() {
        System.out.println("Base");
    }
}

class Derived extends Base {
    public void Print() {
        System.out.println("Derived");
    }
}

public class Main {
    public static void DoPrint(Base o) {
        o.Print();
    }

    public static void main(String[] args) {
        Base x = new Base();
        Base y = new Derived();
        Derived z = new Derived();
        DoPrint(x);
        DoPrint(y);
        DoPrint(z);
    }
}
```

## Output

Base
Derived
Derived

# Polymorphism

```
class Base {
    public void foo() {
        System.out.println("Base");
    }
}

class Derived extends Base {
    private void foo() {
        System.out.println("Derived");
    }
}

public class Main {
    public static void main(String[] args) {
        Base b = new Derived();
        b.foo();
    }
}
```

Output

# Polymorphism

```
class Base {
    public void foo() {
        System.out.println("Base");
    }
}

class Derived extends Base {
    private void foo() {
        System.out.println("Derived");
    }
}

public class Main {
    public static void main(String[] args) {
        Base b = new Derived();
        b.foo();
    }
}
```

## Output

error: foo() in Derived cannot override foo() in Base attempting to assign weaker access privileges; was public

# Polymorphism

Which of these are valid?

1. In Java all classes (excluding Object) inherit from the Object class directly or indirectly. The Object class is root of all classes.
2. Multiple inheritance is not allowed in Java.

Output

# Polymorphism

Which of these are valid?

1. In Java all classes (excluding Object) inherit from the Object class directly or indirectly. The Object class is root of all classes.
2. Multiple inheritance is not allowed in Java.

## Output

1 and 2

# Polymorphism

```
class Grandparent {
    public void Print() {
        System.out.println("Grandparent's Print()");
    }
}

class Parent extends Grandparent {
    public void Print() {
        System.out.println("Parent's Print()");
    }
}

class Child extends Parent {
    public void Print() {
        super.super.Print();
        System.out.println("Child's Print()");
    }
}

public class Main {
    public static void main(String[] args) {
        Child c = new Child();
        c.Print();
    }
}
```

Output

# Polymorphism

```
class Grandparent {
    public void Print() {
        System.out.println("Grandparent's Print()");
    }
}

class Parent extends Grandparent {
    public void Print() {
        System.out.println("Parent's Print()");
    }
}

class Child extends Parent {
    public void Print() {
        super.super.Print();
        System.out.println("Child's Print()");
    }
}

public class Main {
    public static void main(String[] args) {
        Child c = new Child();
        c.Print();
    }
}
```

## Output

error: <identifier> expected
error: not a statement
error: illegal start of expression
error: ';' expected

**Reason:** Cannot call `super.super`

# Polymorphism

```
final class Complex {
    private final double re;
    private final double im;

    public Complex(double re, double im) {
        this.re = re;
        this.im = im;
    }

    public String toString() {
        return "(" + re + " + " + im + "i)";
    }
}

public class Main {
    public static void main(String[] args) {
        Complex c = new Complex(10, 15);
        System.out.println("Complex number is " + c);
    }
}
```

Output

# Polymorphism

```
final class Complex {
    private final double re;
    private final double im;

    public Complex(double re, double im) {
        this.re = re;
        this.im = im;
    }

    public String toString() {
        return "(" + re + " + " + im + "i)";
    }
}

public class Main {
    public static void main(String[] args) {
        Complex c = new Complex(10, 15);
        System.out.println("Complex number is " + c);
    }
}
```

## Output

Complex number is (10.0 + 15.0i)

# Polymorphism

```
class Base {
    int i = 99;

    public void m() {
        System.out.println("Base.m()");
    }

    Base() {
        m();
    }
}

public class Derived extends Base {
    int i = -1;

    public static void main(String[] args) {
        Base b = new Derived();
        System.out.println(b.i);
        b.m();
    }

    public void m() {
        System.out.println("Derived.m()");
    }
}
```

## Output

# Polymorphism

```
class Base {
    int i = 99;

    public void m() {
        System.out.println("Base.m()");
    }

    Base() {
        m();
    }
}

public class Derived extends Base {
    int i = -1;

    public static void main(String[] args) {
        Base b = new Derived();
        System.out.println(b.i);
        b.m();
    }

    public void m() {
        System.out.println("Derived.m()");
    }
}
```

## Output

Derived.m()
99
Derived.m()

# Polymorphism

```
class Base {
    private void method1() {
        System.out.println("Base's method1()");
    }

    public void method2() {
        System.out.println("Base's method2()");
        method1();
    }
}

public class Derived extends Base {
    public void method1() {
        System.out.println("Derived's method1()");
    }

    public static void main(String[] args) {
        Base b = new Derived();
        b.method2();
    }
}
```

## Output

# Polymorphism

```
class Base {
    private void method1() {
        System.out.println("Base's method1()");
    }

    public void method2() {
        System.out.println("Base's method2()");
        method1();
    }
}

public class Derived extends Base {
    public void method1() {
        System.out.println("Derived's method1()");
    }

    public static void main(String[] args) {
        Base b = new Derived();
        b.method2();
    }
}
```

## Output

Base's method2()
Base's method1()

# Polymorphism

```
class X {
    Y b = new Y();

    X() {
        System.out.print("X");
    }
}

class Y {
    Y() {
        System.out.print("Y");
    }
}

public class Z extends X {
    Y y = new Y();

    Z() {
        System.out.print("Z");
    }

    public static void main(String[] args) {
            new Z();
    }
}
```

Output

# Polymorphism

```
class X {
    Y b = new Y();

    X() {
        System.out.print("X");
    }
}

class Y {
    Y() {
        System.out.print("Y");
    }
}

public class Z extends X {
    Y y = new Y();

    Z() {
        System.out.print("Z");
    }

    public static void main(String[] args) {
            new Z();
    }
}
```

## Output

YXYZ

# Numbers

```
public class Main {
    public static void main(String[] args) {
        double p = 1;
        System.out.println(p/0);
    }
}
```

Output

# Numbers

```
public class Main {
    public static void main(String[] args) {
        double p = 1;
        System.out.println(p/0);
    }
}
```

## Output

Infinity

# Numbers

```
public class Main {
    public static void main(String[] args) {
        int p = 1;
        System.out.println(p/0);
    }
}
```

Output

# Numbers

```
public class Main {
    public static void main(String[] args) {
        int p = 1;
        System.out.println(p/0);
    }
}
```

## Output

Exception in thread “main”
java.lang.ArithmeticException:
/ by zero

# Misc

```
public static void main(String[] args) {
    int a[] = {1, 2, 3, 4};
    System.out.print(a instanceof Object);
    /* instanceof checks if something is of a specific type */
}
```

Output

# Misc

```
public static void main(String[] args) {
    int a[] = {1, 2, 3, 4};
    System.out.print(a instanceof Object);
    /* instanceof checks if something is of a specific type */
}
```

## Output

true

# Misc

```
public static void main(String[] args) {
    int a[] = { 1, 2, 053, 4};
    int b[][] = {{1, 2, 4}, {2, 2, 1}, {0, 43, 2}};
    System.out.print(a[3] == b[0][2] );
    System.out.print(" " + (a[2] == b[2][1]));
}
```

## Output

# Misc

```
public static void main(String[] args) {
    int a[] = { 1, 2, 053, 4};
    int b[][] = {{1, 2, 4}, {2, 2, 1}, {0, 43, 2}};
    System.out.print(a[3] == b[0][2] );
    System.out.print(" " + (a[2] == b[2][1]));
}
```

## Output

true true

**Reason:** 0 starts an octal number![3]

[3] https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

# Passing Objects

```
public class Test {
    int a = 1;
    int b = 2;

    Test func(Test obj) {
        Test obj3 = new Test();
        obj3 = obj;
        obj3.a = obj.a++ + ++obj.b;
        obj.b = obj.b;
        return obj3;
    }

    public static void main(String[] args) {
        Test obj1 = new Test();
        Test obj2 = obj1.func(obj1);

        System.out.println(obj1.a + " " + obj1.b);
        System.out.println(obj2.a + " " + obj2.b);
    }
}
```

Output

# Passing Objects

```
public class Test {
    int a = 1;
    int b = 2;

    Test func(Test obj) {
        Test obj3 = new Test();
        obj3 = obj;
        obj3.a = obj.a++ + ++obj.b;
        obj.b = obj.b;
        return obj3;
    }

    public static void main(String[] args) {
        Test obj1 = new Test();
        Test obj2 = obj1.func(obj1);

        System.out.println(obj1.a + " " + obj1.b);
        System.out.println(obj2.a + " " + obj2.b);
    }
}
```

## Output

4 3
4 3

# Passing Objects

```
class Value {
    public int i = 15;
}

public class Test {
    public static void main(String[] args) {
        Test t = new Test();
        t.first();
    }

    public void first() {
        int i = 5;
        Value v = new Value();
        v.i = 25;
        second(v, i);
        System.out.println(v.i);
    }

    public void second(Value v, int i) {
        i = 0;
        v.i = 20;
        Value val = new Value();
        v =  val;
        System.out.println(v.i + " " + i);
    }
}
```

Output

# Passing Objects

```
class Value {
    public int i = 15;
}

public class Test {
    public static void main(String[] args) {
        Test t = new Test();
        t.first();
    }

    public void first() {
        int i = 5;
        Value v = new Value();
        v.i = 25;
        second(v, i);
        System.out.println(v.i);
    }

    public void second(Value v, int i) {
        i = 0;
        v.i = 20;
        Value val = new Value();
        v =  val;
        System.out.println(v.i + " " + i);
    }
}
```

## Output

15 0
20

# Static Keyword

```
public class Main {
    public static void main(String[] args) {
        System.out.println(fun());
    }

    int fun() {
        return 20;
    }
}
```

Output

# Static Keyword

```
public class Main {
    public static void main(String[] args) {
        System.out.println(fun());
    }

    int fun() {
        return 20;
    }
}
```

## Output

error: non-static method fun() cannot be referenced from a static context

# Static Keyword

```
class Base {
    static int x = 0;
}

public class Derived extends Base {
    public static void fun() {
        System.out.println(super.x);
    }

    public static void main(String[] args) {
        fun();
    }
}
```

Output

# Static Keyword

```
class Base {
    static int x = 0;
}

public class Derived extends Base {
    public static void fun() {
        System.out.println(super.x);
    }

    public static void main(String[] args) {
        fun();
    }
}
```

## Output

error: non-static variable super cannot be referenced from a static context

# Static Keyword

```
public class Main {
    public static void main(String[] args) {
        System.out.println(fun());
    }

    static int fun() {
        static int x = 10;
        return x--;
    }
}
```

Output

# Static Keyword

```
public class Main {
    public static void main(String[] args) {
        System.out.println(fun());
    }

    static int fun() {
        static int x = 10;
        return x--;
    }
}
```

## Output

error: illegal start of expression

# Strings

```
String s1 = new String("Test");
String s2 = new String("Test");

if (s1==s2)
    System.out.println("Same");

if (s1.equals(s2))
    System.out.println("Equals");
```

Output

# Strings

```
String s1 = new String("Test");
String s2 = new String("Test");

if (s1==s2)
    System.out.println("Same");

if (s1.equals(s2))
    System.out.println("Equals");
```

## Output

Equals

# Strings

```
String s1 = "Test";
String s2 = "Test";

if (s1==s2)
    System.out.println("Same");

if (s1.equals(s2))
    System.out.println("Equals");
```

## Output

# Strings

```
String s1 = "Test";
String s2 = "Test";

if (s1==s2)
    System.out.println("Same");

if (s1.equals(s2))
    System.out.println("Equals");
```

## Output

Same
Equals

# Strings

```
public class Main {
    public static void main(String[] args) {
        String s = new String("Java");
        s.concat(" SE 8");
        s.replace('8','9');
        System.out.print(s);
    }
}
```

Output

# Strings

```
public class Main {
    public static void main(String[] args) {
        String s = new String("Java");
        s.concat(" SE 8");
        s.replace('8','9');
        System.out.print(s);
    }
}
```

## Output

Java

**Reason:** `concat` and `replace` return the modified `String`. So discarded because not used. Also Strings are immutable in Java.

# Misc

```
public class Main {
    public static void main(String[] args) {
        for(int i = 0; i < 3; i++) {
            System.out.print(i);
        }
        System.out.print(i);
    }
}
```

Output

# Misc

```
public class Main {
    public static void main(String[] args) {
        for(int i = 0; i < 3; i++) {
            System.out.print(i);
        }
        System.out.print(i);
    }
}
```

## Output

Test.java:9: error: cannot find symbol

# Misc

```
int []a = {1,2,3,4,5,6};
int i = a.length - 1;

while(i >= 0) {
    System.out.print(a[i]);
    i--;
}
```

## Output

# Misc

```
int []a = {1,2,3,4,5,6};
int i = a.length - 1;

while(i >= 0) {
    System.out.print(a[i]);
    i--;
}
```

## Output

654321

# Misc

```
public static void main(String[] args) {
    int x = 0;
    int y = 10;

    try {
        y /= x;
    }
    System.out.println("/ by 0");
    catch(Exception e) {
        System.out.print("error");
    }
}
```

Output

# Misc

```
public static void main(String[] args) {
    int x = 0;
    int y = 10;

    try {
        y /= x;
    }
    System.out.println("/ by 0");
    catch(Exception e) {
        System.out.print("error");
    }
}
```

## Output

error: 'try' without 'catch', 'finally' or resource declarations
error: 'catch' without 'try'

# Misc

```
public class Main {
    void myMethod(int i) {
        System.out.println("int version");
    }

    void myMethod(String s) {
        System.out.println("String version");
    }

    public static void main(String[] args) {
        Main obj = new Main();
        char ch = 'c';
        obj.myMethod(ch);
    }
}
```

Output

# Misc

```
public class Main {
    void myMethod(int i) {
        System.out.println("int version");
    }

    void myMethod(String s) {
        System.out.println("String version");
    }

    public static void main(String[] args) {
        Main obj = new Main();
        char ch = 'c';
        obj.myMethod(ch);
    }
}
```

## Output

int version

**Reason:** Widening of argument input (primitive type).
[4]

[4] https://docs.oracle.com/javase/specs/jls/se8/html/jls-5.html

# Last Exercise Session

```
public class Base {
    int a = 10;

    public static void main(String[] args) {
        new Base().print();
    }

    public void print() {
        int a = 8;
        System.out.print(a + " ");
    }
}
```

Output:

# Last Exercise Session

```
public class Base {
    int a = 10;

    public static void main(String[] args) {
        new Base().print();
    }

    public void print() {
        int a = 8;
        System.out.print(a + " ");
    }
}
```

## Output:

8

**Reason:** Variable shadowing.

# Last Exercise Session

```
public class Base {
    public static void main(String[] args) {
        new Base().pass();
    }

    public void pass() {
        int a = 10, b = 20;
        print(a);
    }

    public void print(int a) {
        int c = b/a;
        System.out.print(c);
    }
}
```

Output:

# Last Exercise Session

```
public class Base {
    public static void main(String[] args) {
        new Base().pass();
    }

    public void pass() {
        int a = 10, b = 20;
        print(a);
    }

    public void print(int a) {
        int c = b/a;
        System.out.print(c);
    }
}
```

## Output:

/Base.java:13: error: cannot find symbol
int c = b/a;

# Last Exercise Session

```
public class Base {
    static int x = 10;

    public static void main(String[] args) {
        for(int x = 0; x < 5; x++) {}
        System.out.print(x);
    }
}
```

Output:

# Last Exercise Session

```
public class Base {
    static int x = 10;

    public static void main(String[] args) {
        for(int x = 0; x < 5; x++) {}
        System.out.print(x);
    }
}
```

## Output:

10

**Reason:** Variable shadowing only within loop.

# Last Exercise Session

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}

public class Main {
    public static void main(String[] args) {
        /* Initialize OuterClass */
        /* Initialize InnerClass */
        /* Initialize StaticNestedClass */
    }
}
```

Output:

# Last Exercise Session

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}

public class Main {
    public static void main(String[] args) {
        /* Initialize OuterClass */
        /* Initialize InnerClass */
        /* Initialize StaticNestedClass */
    }
}
```

## Output:

```
OuterClass oc = new
OuterClass;
```

5

[5] https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

# Last Exercise Session

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}

public class Main {
    public static void main(String[] args) {
        /* Initialize OuterClass */
        /* Initialize InnerClass */
        /* Initialize StaticNestedClass */
    }
}
```

## Output:

```
OuterClass oc = new
OuterClass;
OuterClass.InnerClass ic =
oc.new InnerClass();
```

5

[5] https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

# Last Exercise Session

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}

public class Main {
    public static void main(String[] args) {
        /* Initialize OuterClass */
        /* Initialize InnerClass */
        /* Initialize StaticNestedClass */
    }
}
```

## Output:

```
OuterClass oc = new
OuterClass;
```

```
OuterClass.InnerClass ic =
oc.new InnerClass();
```

```
OuterClass.StaticNestedClass
snc = new
OuterClass.StaticNestedClass();
```

5

[5] https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

# Last Exercise Session

```
public class Outer {
    public int a = 1;
    private int b = 2;

    public void method(final int c) {
        int d = 3;
        if (c % 2 == 1) {
            d = 4;
        }

        class Inner {
            private void iMethod(int e) {
                /* What variables can we access here? */
            }
        }
    }
}
```

Output:

# Last Exercise Session

```
public class Outer {
    public int a = 1;
    private int b = 2;

    public void method(final int c) {
        int d = 3;
        if (c % 2 == 1) {
            d = 4;
        }

        class Inner {
            private void iMethod(int e) {
                /* What variables can we access here? */
            }
        }
    }
}
```

## Output:

a b c e

*Note:* Only (effectively) final variables/fields are accessible like this starting Java 8. [5]

[5] https://docs.oracle.com/javase/tutorial/java/javaOO/localclasses.html

# Last Exercise Session

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
        /* How to access OuterClass? */
    }
}

public class Main {
    public static void main(String[] args) {
    }
}
```

Output:

# Last Exercise Session

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
        /* How to access OuterClass? */
    }
}

public class Main {
    public static void main(String[] args) {
    }
}
```

## Output:

`OuterClass.this`

# Last Exercise Session

```
public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<Integer>();

        for (int i = 0; i < list.size(); i++) {
            list.add(i, 42);
        }
        System.out.println(list);
    }
}
```

Output:

# Last Exercise Session

```
public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<Integer>();

        for (int i = 0; i < list.size(); i++) {
            list.add(i, 42);
        }
        System.out.println(list);
    }
}
```

## Output:

No output

**Reason:** Loop never terminates.

# Last Exercise Session

```
public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        /* Read in numbers */
        filterEvens(numbers);
        System.out.println(numbers);
    }

    private void filterEvens(ArrayList<Integer> list) {
        for (int iter : list) {
            if (iter % 2 == 0) {
                list.remove(iter);
            }
        }
    }
}
```

Output:

# Last Exercise Session

```
public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        /* Read in numbers */
        filterEvens(numbers);
        System.out.println(numbers);
    }

    private void filterEvens(ArrayList<Integer> list) {
        for (int iter : list) {
            if (iter % 2 == 0) {
                list.remove(iter);
            }
        }
    }
}
```

## Output:

Exception in thread “main” java.util.ConcurrentModificationException.
Also semantically incorrect!

**Reason:** Invalid to modify the data structure when iterating like this.

# Last Exercise Session

```
public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        /* Read in numbers */
        filterEvens(numbers);
        System.out.println(numbers);
    }

    private void filterEvens(ArrayList<Integer> list) {
        for (Integer iter : list) {
            if (iter.intValue() % 2 == 0) {
                list.remove(iter);
            }
        }
    }
}
```

Output:

# Last Exercise Session

```
public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        /* Read in numbers */
        filterEvens(numbers);
        System.out.println(numbers);
    }

    private void filterEvens(ArrayList<Integer> list) {
        for (Integer iter : list) {
            if (iter.intValue() % 2 == 0) {
                list.remove(iter);
            }
        }
    }
}
```

## Output:

Exception in thread “main”
java.util.ConcurrentModificationException

**Reason:** Invalid to modify the data structure when iterating like this.

# Last Exercise Session

```
public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        /* Read in numbers */
        filterEvens(numbers);
        System.out.println(numbers);
    }

    private void filterEvens(ArrayList<Integer> list) {
        for (Iterator<Integer> iter = list.iterator();
     iter.hasNext(); ) {
            if (iter.next() % 2 == 0) {
                iter.remove();
            }
        }
    }
}
```

Output:

# Last Exercise Session

```
public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        /* Read in numbers */
        filterEvens(numbers);
        System.out.println(numbers);
    }

    private void filterEvens(ArrayList<Integer> list) {
        for (Iterator<Integer> iter = list.iterator();
     iter.hasNext(); ) {
            if (iter.next() % 2 == 0) {
                iter.remove();
            }
        }
    }
}
```

## Output:

Returns expected result

# Last Exercise Session

```
public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        /* Read in numbers */
        filterEvens(numbers);
        System.out.println(numbers);
    }

    private void filterEvens(ArrayList<Integer> list) {
        for (int i = list.size() - 1; i >= 0; i--) {
            if (list.get(i) % 2 == 0) {
                    list.remove(i);
            }
        }
    }
}
```

Output:

# Last Exercise Session

```
public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        /* Read in numbers */
        filterEvens(numbers);
        System.out.println(numbers);
    }

    private void filterEvens(ArrayList<Integer> list) {
        for (int i = list.size() - 1; i >= 0; i--) {
            if (list.get(i) % 2 == 0) {
                    list.remove(i);
            }
        }
    }
}
```

## Output:

Returns expected result

# Last Exercise Session

```
public class Main {
    static int size = 10;
    static Boolean[] bools = new Boolean[size];

    public static void main(String args[]) {
        for (int i = 0; i < size; i += 2) {
            bools[i] = new Boolean(Boolean.TRUE);
        }

        int i = 0;
        do {
            System.out.println("Cell[" + i + "]: " + bools[i]);
        } while (bools[i++]);
    }
}
```

Output:

# Last Exercise Session

```
public class Main {
    static int size = 10;
    static Boolean[] bools = new Boolean[size];

    public static void main(String args[]) {
        for (int i = 0; i < size; i += 2) {
            bools[i] = new Boolean(Boolean.TRUE);
        }

        int i = 0;
        do {
            System.out.println("Cell[" + i + "]: " + bools[i]);
        } while (bools[i++]);
    }
}
```

## Output:

Exception in thread “main” Cell[0]: true
Cell[1]: null
java.lang.NullPointerException
    at Main.main(Main.java:13)

**Reason:** Box will automatically be unboxed (Boolean is null for all odd indices).

# Last Exercise Session

```
public class Super {
    public Super() {
        init();
    }
    public void init() {
    }
}

public class Sub extends Super {
    String s;
    int length;
    public Sub(String s) {
        this.s = (s == null) ? "" : s;
    }
    public void init() {
        length = s.length();
    }
}

public class Main{
    public static void main(String[] args) {
        Sub s = new Sub("test");
        System.out.println(s);
    }
}
```

Output:

# Last Exercise Session

```java
public class Super {
    public Super() {
        init();
    }
    public void init() {
    }
}

public class Sub extends Super {
    String s;
    int length;
    public Sub(String s) {
        this.s = (s == null) ? "" : s;
    }
    public void init() {
        length = s.length();
    }
}

public class Main{
    public static void main(String[] args) {
        Sub s = new Sub("test");
        System.out.println(s);
    }
}
```

## Output:

NullPointerException

**Reason:** Non final methods in constructors [5]

[5] https://help.semmle.com/wiki/display/JAVA/Non-final+method+invocation+in+constructor

# Last Exercise Session

```
int one = '1';
int char_i = 'i';

System.out.println("120? " + one + '2' + char_i);
```

Output:

# Last Exercise Session

```
int one = '1';
int char_i = 'i';

System.out.println("120? " + one + '2' + char_i);
```

## Output:

120? 492105

**Reason:** Variable type defines `toString()` call via Wrapper object. [5]

[5] https://en.wikibooks.org/wiki/Java_Programming/Literals

# Last Exercise Session

```
boolean a = true;
boolean b = false;
boolean c = true;

if (a == true)
    if (b == true) {
        if (c == true) {
            System.out.println("Some things are true in this
     world");
        } else {
            System.out.println("Nothing is true in this world!");
        }
    } else if (a && (b = c)) {
        System.out.println("It's too confusing to tell what is
     true and what is false");
    } else {
        System.out.println("Hey this won't compile");
    }
}
```

Output:

# Last Exercise Session

```
boolean a = true;
boolean b = false;
boolean c = true;

if (a == true)
    if (b == true) {
        if (c == true) {
            System.out.println("Some things are true in this
     world");
        } else {
            System.out.println("Nothing is true in this world!");
        }
    } else if (a && (b = c)) {
        System.out.println("It's too confusing to tell what is
     true and what is false");
    } else {
        System.out.println("Hey this won't compile");
    }
}
```

## Output:

It's too confusing to tell what is true and what is false

**Reason:** Assignment of existing variables possible in conditional check. [5]

[5] https://stackoverflow.com/questions/16148580/assign-variable-value-inside-if-statement

# Last Exercise Session

```
public class Ternary
{
    public static void main(String args[])
    {
        int a = 5;
        System.out.println("Value is - " + ((a < 5) ? 9.9 : 9));
    }
}
```

Output:

# Last Exercise Session

```
public class Ternary
{
    public static void main(String args[])
    {
        int a = 5;
        System.out.println("Value is - " + ((a < 5) ? 9.9 : 9));
    }
}
```

## Output:

9.**0**

**Reason:** Widening of integer to double to match common return type of ternary.

# Last Exercise Session

```java
// Filename; SuperclassX.java
package packageX;
public class SuperclassX {
    int superclassVarX;

    protected void superclassMethodX() {}
}

// Filename SubclassY.java
package packageX.packageY;
public class SubclassY extends SuperclassX {
    SuperclassX objX = new SubclassY();
    SubclassY objY = new SubclassY();

    void subclassMethodY() {
        int i = 0;
        objY.superclassMethodX();
        i = objY.superclassVarX;
    }
}
```

Output:

# Last Exercise Session

```java
// Filename; SuperclassX.java
package packageX;
public class SuperclassX {
    int superclassVarX;

    protected void superclassMethodX() {}
}

// Filename SubclassY.java
package packageX.packageY;
public class SubclassY extends SuperclassX {
    SuperclassX objX = new SubclassY();
    SubclassY objY = new SubclassY();

    void subclassMethodY() {
        int i = 0;
        objY.superclassMethodX();
        i = objY.superclassVarX;
    }
}
```

## Output:

Compile Error when accessing field of `SuperclassX`

**Reason:** The field is not protected but only visible within the package.

# Last Exercise Session

```
int Output = 10;
boolean b1 = false;
if((b1 == true) && ((Output += 10) == 20)) {
       System.out.println("We are equal " + Output);
} else {
        System.out.println("Not equal! " + Output);
}
```

Output:

# Last Exercise Session

```
int Output = 10;
boolean b1 = false;
if((b1 == true) && ((Output += 10) == 20)) {
       System.out.println("We are equal " + Output);
} else {
        System.out.println("Not equal! " + Output);
}
```

## Output:

Not equal! 10

**Reason:** && evaluates left to right, finds false and stops further evaluation immediately.

# Last Exercise Session

```
public class MyClass {
    static String s1 = "I am unique!";

    public static void main(String args[]) {
        String s2 = "I am unique!";
        String s3 = new String(s1);
        System.out.println(s1 == s2);
        System.out.println(s1.equals(s2));
        System.out.println(s3 == s1);
        System.out.println(s3.equals(s1));
        System.out.println(TestClass.s4 == s1);
    }
}

class TestClass {
    static String s4 = "I am unique!";
}
```

Output:

# Last Exercise Session

```
public class MyClass {
    static String s1 = "I am unique!";

    public static void main(String args[]) {
        String s2 = "I am unique!";
        String s3 = new String(s1);
        System.out.println(s1 == s2);
        System.out.println(s1.equals(s2));
        System.out.println(s3 == s1);
        System.out.println(s3.equals(s1));
        System.out.println(TestClass.s4 == s1);
    }
}

class TestClass {
    static String s4 = "I am unique!";
}
```

## Output:

true
true
false
true
true

# Last Exercise Session

```
class A {
    public A(int a) {
        System.out.println("a");
    }
}

class B extends A {
    public B(int a) {
        System.out.println("b");
        super(a);
    }
}
```

Output:

# Last Exercise Session

```
class A {
    public A(int a) {
        System.out.println("a");
    }
}

class B extends A {
    public B(int a) {
        System.out.println("b");
        super(a);
    }
}
```

## Output:

Compile-time error.

**Reason:** `super(a)` must be called first.

# Last Exercise Session

```
class Test {
    int i;
}

class Main {
    public static void main(String args[]) {
        Test t;
        System.out.println(t.i);
    }
}
```

Output:

# Last Exercise Session

```
class Test {
    int i;
}

class Main {
    public static void main(String args[]) {
        Test t;
        System.out.println(t.i);
    }
}
```

## Output:

Compile-time error.

**Reason:** Use before definition. All local vars must be initialised first!

# Last Exercise Session

```
class Test {
    int i;
}

class Main {
    public static void main(String args[]) {
        Test t = new Test();
        System.out.println(t.i);
    }
}
```

Output:

# Last Exercise Session

```
class Test {
    int i;
}

class Main {
    public static void main(String args[]) {
        Test t = new Test();
        System.out.println(t.i);
    }
}
```

## Output:

0

**Reason:** Global field value is 0 initialised.

# Last Exercise Session

```
class demoClass {
    int a = 1;
    void func() {
        demo obj = new demo();
        obj.display();
    }
    class demo {
        int b = 2;

        void display() {
            System.out.println("\na = " + a);
        }
    }
    void get() {
        System.out.println("\nb = " + b);
    }
}

class Test {
    public static void main(String[] args) {
        demoClass obj = new demoClass();
        obj.func();
        obj.get();
    }
}
```

Output:

# Last Exercise Session

```
class demoClass {
    int a = 1;
    void func() {
        demo obj = new demo();
        obj.display();
    }
    class demo {
        int b = 2;

        void display() {
            System.out.println("\na = " + a);
        }
    }
    void get() {
        System.out.println("\nb = " + b);
    }
}

class Test {
    public static void main(String[] args) {
        demoClass obj = new demoClass();
        obj.func();
        obj.get();
    }
}
```

## Output:

Compile-time error.

**Reason:** Cannot access field of inner class like this. Would be possible with `new demo().b` instead.

# Last Exercise Session

```
class First {
    void display() {
        System.out.println("Inside First");
    }
}

class Second extends First {
    void display() {
        System.out.println("Inside Second");
    }
}

class Test {
    public static void main(String[] args) {
        First obj1 =  new First();
        Second obj2 =  new Second();

        First ref;
        ref = obj1;
        ref.display();

        ref = obj2;
        ref.display();
    }
}
```

Output:

# Last Exercise Session

```
class First {
    void display() {
        System.out.println("Inside First");
    }
}

class Second extends First {
    void display() {
        System.out.println("Inside Second");
    }
}

class Test {
    public static void main(String[] args) {
        First obj1 =  new First();
        Second obj2 =  new Second();

        First ref;
        ref = obj1;
        ref.display();

        ref = obj2;
        ref.display();
    }
}
```

## Output:

Inside First
Inside Second

**Reason:** `ref` is reference variable of class `First`, so runtime polymorphism applies.

# Last Exercise Session

```
class A {
    public A(int a) {
        System.out.println("a");
    }
}

class B extends A {
    public B(int a) {
        System.out.println("b");
    }
}
```

Output:

# Last Exercise Session

```
class A {
    public A(int a) {
        System.out.println("a");
    }
}

class B extends A {
    public B(int a) {
        System.out.println("b");
    }
}
```

## Output:

Compile-time error.

**Reason:** `super(a)` not called.

# Quiz Sources

The quiz material presented is based and extended from these quizzes (in no particular order):

- http://www.gocertify.com/quizzes/java/scjp1.html
- http://www.gocertify.com/quizzes/oracle/ocajp-java-quiz-1.html
- http://www.gocertify.com/quizzes/java/scjp2.html
- http://www.geeksforgeeks.org/java-gq/inheritance-2-gq/
- http://www.geeksforgeeks.org/java-class-and-object-question-6/
- http://www.geeksforgeeks.org/static-keyword-in-java/
- http://www.geeksforgeeks.org/java-gq/class-and-object-2-gq/
- http://ce.sharif.edu/courses/92-93/2/ce244-1/resources/root/Quizes/Quiz5_92932.pdf

# Maybe Interesting

Here I list some links that may be interesting (no guarantee on correctness, validity, nor requirement on the exam):

- https://vimeo.com/7403673 (Video about text, numbers, dates in programming; watch it, you will learn some stuff :))
- http://www.lst.inf.ethz.ch/education/einfuehrung-in-die-programmierung-i--252-0027-.html (Lecture Website)
- https://softwareengineering.stackexchange.com/questions/258050/why-overriding-a-static-method-does-not-result-in-polymorphism-in-java
- https://www.ntu.edu.sg/home/ehchua/programming/java/J3b_OOPInheritancePolymorphism.html
- https://stackoverflow.com/questions/205239/why-can-final-constants-in-java-be-overridden
- https://stackoverflow.com/questions/7486012/static-classes-in-java

# Some More Quizzes

Some more quizzes (no guarantee on correctness, quality nor requirement on the exam):

- `https://www.tacoma.uw.edu/institute-technology/java-programming-self-assessment`
- `https://courses.cs.washington.edu/courses/cse373/17au/#exams` (Data Structures & Algorithms)
- `https://www.journaldev.com/15161/core-java-quiz`
- `http://cnds.eecs.jacobs-university.de/courses/java-2015/`
- `http://education.oracle.com/education/downloads/Exam803_SampleQuestion.pdf`
- `http://education.oracle.com/education/downloads/Exam808_SampleQuestion.pdf`