TECHNISCHE
UNIVERSITÄT
WIEN

Inheritance
Interfaces

Subtyping

```
class A { }

class B extends A { }

class C { }

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

        A v1 = new A();          // Compiler errors?
        B v2 = new A();          // Compiler errors?
        A v3 = new B();          // Compiler errors?
        B v4 = new C();          // Compiler errors?
        Object v5 = new C();     // Compiler errors?
        Object v6 = new B();     // Compiler errors?

    }
}
```

Subtyping

```
class A { }

class B extends A { }

class C { }

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

        A v1 = new A();          // OK
        B v2 = new A();          // Error: A is not a B
        A v3 = new B();          // OK
        B v4 = new C();          // Error: C is not a B
        Object v5 = new C();     // OK
        Object v6 = new B();     // OK

    }
}
```

Inheritance - Constructors

```
class A {
    public A() {
        super();   // Compiler error? What is called here?
        System.out.println("A here!");
    }
}

class B extends A {
    public B() {
        this();   // Compiler error? What is called here?
    }
}

public class Main {
    public static void main(String[] args) {
        A v1 = new A();   // Output?
        B v2 = new B();   // Output?
    }
}
```

Inheritance - Constructors

```
class A {
    public A() {
        super();   // OK: Default constructor of Object is called
        System.out.println("A here!");
    }
}

class B extends A {
    public B() {
        this();   // Error: Recursive constructor call
    }
}

public class Main {
    public static void main(String[] args) {
        A v1 = new A();   // Output: A here!
        B v2 = new B();   // Output: A here!
    }
}
```

Method Overriding

```
class A {
    public void x() {  System.out.println("A:x"); }
    public void y() {  System.out.println("A:y"); }
}

class B extends A {
    public void y() {  System.out.println("B:y"); }
}

public class Main {
    public static void main(String[] args) {
        A v1 = new A();
        A v2 = new B();

        v1.x();   // Output?
        v1.y();   // Output?
        v2.x();   // Output?
        v2.y();   // Output?
    }
}
```

Method Overriding

```
class A {
    public void x() {  System.out.println("A:x"); }
    public void y() {  System.out.println("A:y"); }
}

class B extends A {
    @Override   // Use @Override!
    public void y() {  System.out.println("B:y"); }
}

public class Main {
    public static void main(String[] args) {
        A v1 = new A();
        A v2 = new B();

        v1.x();   // Output: A:x
        v1.y();   // Output: A:y
        v2.x();   // Output: A:x
        v2.y();   // Output: B:y
    }
}
```

Abstracts

```
// Any errors here?

abstract class A {
    public abstract void x();
}

class B extends A {
    @Override
    public void x() { }
}

class C { } extends A { }

public class Main {
    public static void main(String[] args) {
        A v1 = new A();
        B v2 = new B();
        C v3 = new C();
    }
}
```

Abstracts

```
abstract class A {
    public abstract void x();
}

class B extends A {
    @Override
    public void x() { }
}

class C { } extends A { }  // Error: C is not abstract or does not
                           // implement x()

public class Main {
    public static void main(String[] args) {
        A v1 = new A();  // Error: A cannot be instantiated
        B v2 = new B();
        C v3 = new C();
    }
}
```

Interfaces

- Interfaces are similar to abstract classes
- All members are `public`
- All methods are `abstract`*
- Only `final static` fields (constants) allowed
- Classes can implement multiple interfaces

* There are special rules regarding default methods. We are not considering them here.
See https://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html

Interfaces - Example

```
interface A {
    void a();
}

interface B {
    void b();
}

class C implements A, B {
    @Override
    void a() { }

    @Override
    void b() { }
}
```

Reading Code - `ArrayList<E>`

- What are the implemented interfaces and super classes of `ArrayList<E>`?
- What happens when `new ArrayList<int>(20)` is called?

Exercise: Iterable

```
class IterableNumbers implements java.lang.Iterable<Integer> {
    private Integer[] numbers;

    public IterableNumbers(Integer[] numbers) {
        this.numbers = numbers;
    }
    ...
}

public class Main {
    public static void main(String[] args) {
        IterableNumbers n = new IterableNumbers(new Integer[] {1, 2, 3});
        for (Integer i : n) {
            System.out.println(i);
        }
    }
}
```