Methods

• Constructors
• Instance methods
• Class methods
Constructors

- have the same name as the class
- do not return anything
- they begin with a call to the constructor from the parent class (super(...))

```java
public class MyClass{
    public MyClass(){}
    ...
    MyClass mc = MyClass();
}
```
Instance Method

- are called on a reference to an instance
- can return values
- C-style declaration and value return

```java
public class MyClass{
    public boolean test(){
        return true;
    }
}
...
MyClass mc=new MyClass();
mc.test();
```
Class Method

• Are called on the class name, instance names
• declared static

```java
public class MyClass{
    public static boolean test(){
        return true;
    }
}
```

`MyClass.test();`
The main method

- static, returns nothing, public, String[] as parameters

```java
public class MyClass{
    public static void main(String[] args){...
    }
}
```
Inheritance

• Methods and fields are inherited
• They can be used in a child class without any redeclaration
• They can be overridden
• Overloading is ok
Exercises

• final & volatile?
• static and multithreaded?
• local variable usable from another class?
• declare a String that would be visible from outside the package, shared by all instances and can be accessed concurrently.
Why doesn’t it compile?

```java
public class HelloWorld {
    public String message="Hello World!";
    public HelloWorld(String s){
        super();
        message=s;
    }
}
public class HelloWorld2 extends HelloWorld{
    public HelloWorld2(String s){
        message=s;
    }
}
```
Why doesn’t it compile?

```java
class HelloWorld {
    public String message="Hello World!";
    private HelloWorld(String s){
        super();
        message=s;
    }
}

class HelloWorld2 extends HelloWorld{
    public HelloWorld2(String s){
        super(s);
    }
}
```
What happens here?

public class Parent {
    public Parent wave(Parent p){
        System.out.println("in the parent");
        return null;
    }
}
public class Child extends Parent{
    public Child wave(Object o){
        System.out.println("in the child");
        return null;
    }
}
public static void main(String [] args){
    Parent p=new Child();
    p.wave(p);
}
...and here?

```java
public class Parent {
    public Parent wave(Parent p){
        System.out.println("in the parent");
        return null;
    }
}

public class Child extends Parent{
    public Child wave(Parent p){
        System.out.println("in the child");
        return null;
    }

    public static void main(String []args){
        Parent p=new Child();
        p.wave(p);
    }
}
```
public class HelloWorld{
    String s="Hello World";
    public static void main(String []args){
        System.out.println(s);
    }
}
Expressions

- Method Invocation
- instanceof
- Casts
- blocks
- control-flow structures
Invoking Methods

• Already saw that:

```java
MyClass mc = new MyClass();
mc.test();
MyClass.testStatic();
this.test();
test();
```
instanceof

• instruction from the language
• Returns a boolean

mc instanceof MyClass
Casts

• Used to assign an instance of a super class to a variable of a subclass

• May fail at runtime!

    String S = (String)myObject;
Blocks

• As in many languages blocks can be declared in place of any instruction and define a name space:

```java
{
    ...
}
```
if

- if (test) block/instruction;
- if (test) block/instruction; else block/instruction;

```java
if (ms instanceof MyClass)
    if (!(ms instanceof String))
        System.out.println("MyClass not String");
    else
        System.out.println("not MyClass");
```
switch

switch(expression){
    case constant: {}
    ...
    default: {}
}

while

while (test){
    ...
}

do...while

do{
...
}
while (test)
for (init; test; loopInst) {
    ...
}

Enhanced for

• if the expression is Iterable (arrays are...)

```java
for (Type varName: expression) {
    ...
}
```
break/continue

• break breaks the innermost loop
• continue goes back to its start
• with labels: act on the labeled structure