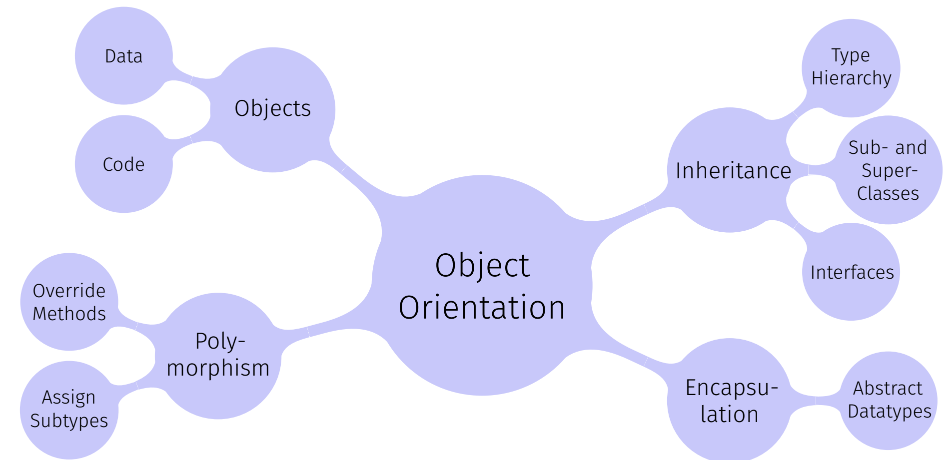
15. Java Object Orientation II

Polymorphism

Object Orientation: Different Aspects



Polymorphism

- **Override Methods:** Inherited methods from a superclass can be overriden: Same signature, new code.
- **Variable Assignment:** Objects of a given type can be assigned to variables of any supertype.

Overriding Methods

Inherited methods of a supertype can get a new implementation. Same **signature**, new **code**.
We remember the method **alarm()** for last time. This abstract method was defined in class **Wind** as follows

```
class Wind extends Measurement {
    int speed;
    ...
    boolean alarm(){  // implements abstract method alarm()
        return this.speed > 80;
    }
}
```

Overriding Methods

We define a new subclass **WindWithGusts**, that also tracks **gusts** in addition to windspeed and direction.

```
/*
 * Fancy windsensor that tracks gusts. Requires special hardware.
 */
class WindWithGusts extends Wind {
    int gusts;

    @Override
    boolean alarm(){  // replaces implementation of supertype
        return this.speed > 80 || this.gusts > 20;
    }
}
```

Inherited from **Wind** (↑ this.speed)

Access to Overriden Method: super Keyword

A subclass doesn't have to repeat the code that is being overridden.
⇒ Call of the overriden implementation using keyword super, **but only within the overriding implementation**

```
class WindWithGusts extends Wind {
    int gusts;

    @Override
    boolean alarm(){  // replaces implementation of supertype
        return super.alarm() || this.gusts > 20;
    }
}
```

Executes: this.speed > 80 ; (↑ super.alarm())

Access to Constructors of Superclass

Setting: Creation of a measurement always requires a coordinate.
→ Constructor in class **Measurement**

```
class Measurement {
    Coordinate position;

    Measurement(float lat, float lon){
        this.position = new Coordinate(lat, lon);
    }
    ...
}
```


Access to Constructors of Superclass

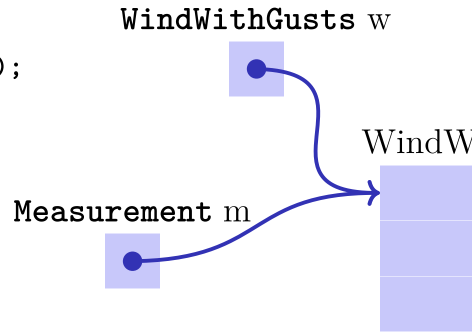
- Using keyword super, a constructer of a superclass can be called.
- The amount and types of the arguments determines **which** constructer will be called
- Calling super(...) **must** be the first instruction!

```
class Wind {
    ...
    Wind(float lat, float lon, int speed, int direction ){
        super(lat, lon);
        this.speed = speed;
        this.direction = direction;
    }
    ...
}
```

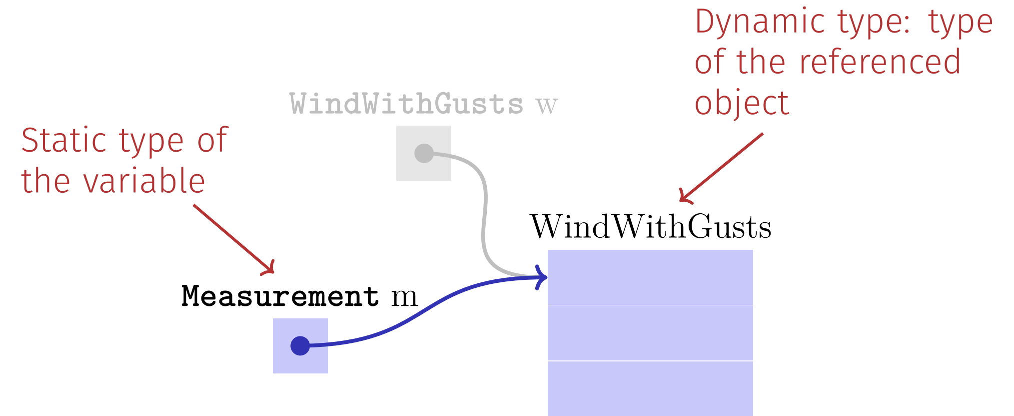
Polymorphic References

Variables of a declared type can reference objects of a subtype.

```
WindWithGusts w = new WindWithGusts();

Measurement m;

m = w; // polymorphic reference!



// But this doesn't compile: w = m;
```



Static vs. Dynamic Type



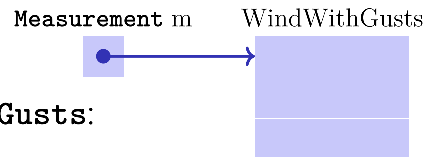
Dynamic Methodbinding

When calling a method, the implementation of the **dynamic type** is executed!

Call:

```
m.alarm();
```

⇒ Executed code from class **WindWithGusts**:




```
@Override
boolean alarm(){
    return super.alarm() || this.gusts > 20;
}
```


Usages for dynamic binding

Given: A list of different kinds of measurements (**Temperatures, Wind, ...**
Wanted: A list of measurements that cause an alarm.

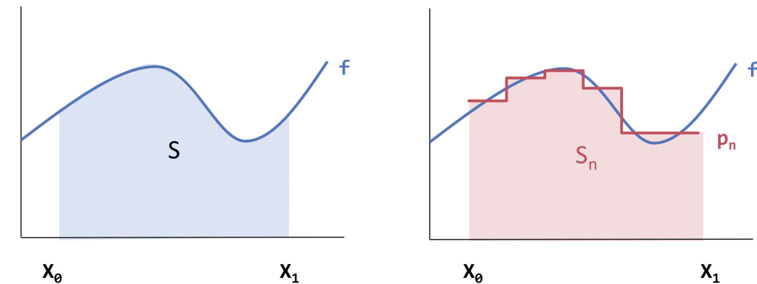
```
void filterByAlarm(Measurement[] measurements){
    for (int i = 0; i < measurements.length; ++i){
        if (measurements[i].alarm()){   //dynamic method binding!
            measurements[i]=null;       //remove from array
        }
    }
}
```

Second Example: Numerical Integration

Goal: Implement a software framework for numerical integration of an arbitrary function $f : \mathbb{R} \to \mathbb{R}$
Problem: How do we represent the function in a generic way? (Java does not offer variables of function type)
Answer: we use inheritance and polymorphism.

Numerical Integration

Approximation of the integral of a function f in an integration interval $[x_0, x_1]$
Simplest possible approach: approximate f with a piecewise constant function p_n with n pieces.



Integrate $x \mapsto x^2$

```
public double Integrate(double x0, double x1, int n){
  double sum = 0;
  double width = (x1-x0)/n; // interval width
  for (int i = 0; i<n; ++i){
    double x = x0 + i*width + 0.5*width; // mid of interval
    double y = x*x; // function value
    sum += y * width; // rectangle area
  }
  return sum;
}
```

$$\int_{x_0}^{x_1} x^2 \, \mathrm{d}x$$

Functions

Generic

```
public abstract class Function {
  public abstract double evaluate(double x);
}
```

$x \mapsto x^2$

```
public class Square extends Function {
  @Override
  public double evaluate(double x){
    return x*x;
  }
}
```

Normal Distribution Probability Density

$x \mapsto \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{\frac{-(x-\mu)^2}{2\sigma^2}}$

```
class Normal extends Function{
  double mu;
  double sigma;
  Normal(double m, double s)
    mu = m; sigma = s;
  }

  @Override
  public double evaluate(double x) {
    return 1/Math.sqrt(2*Math.PI)/sigma
        * Math.exp(-(x-mu)*(x-mu)/(2*sigma*sigma));
  }
}
```


Integrate (generic)

```
public double Integrate(Function f, double x0, double x1, int n){
  double sum = 0;
  double width = (x1-x0)/n;
  for (int i = 0; i<n; ++i){
    double x = x_0 + i*width + 0.5*width;
    double y = f.evaluate(x); // evaluate of the dynamic type
    sum += y * width;
  }
  return sum;
}
```

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x$$

Application

```
Function sq = new Square();
Out.println(Integrate(sq,0,2,1000); // 2.667

Function normal = new Normal(0,1);
Out.println(Integrate(normal,-3,3,1000); // 0.997
```


Conclusion of the Concepts

of Object Oriented Programming

Encapsulation, Information Hiding

- hiding the state and implementation details of an object
- definition of an interface for access to the internal data structures $\rightarrow$ abstraction
- makes the assertion of invariants possible

Conclusion of the Concepts

of Object Oriented Programming

Inheritance

- objects can inherit properties from objects
- derived objects can provide new properties or overwrite existing ones
- makes the reuse of code and data possible

Conclusion of the Concepts

of Object Oriented Programming

Polymorphism

- a designator can take on different data types depending on its use
- different data types can react differently when their common interface is accessed in the same way
- makes it possible to extend libraries in a non-invasive way