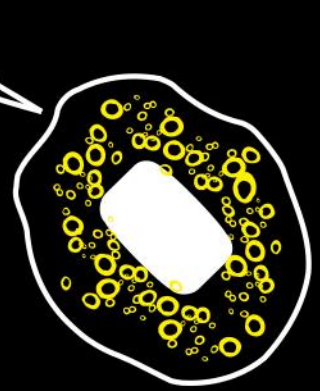
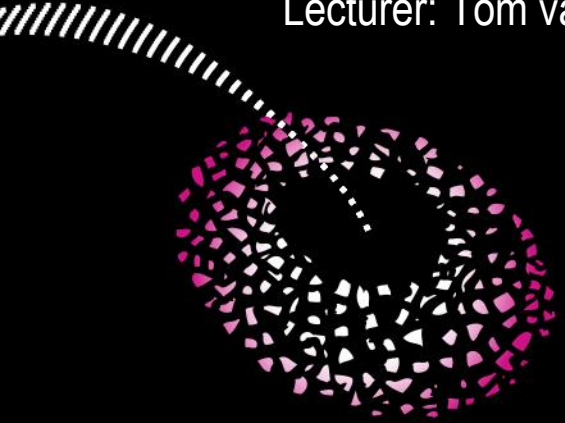
UNIVERSITY OF TWENTE.

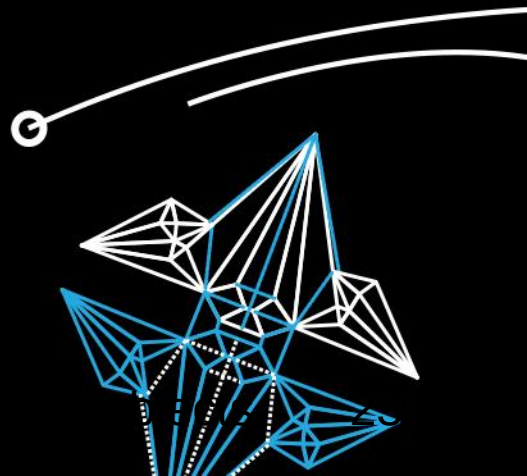


Java subtypes

Topic of Software Systems (TCS module 2)

Lecturer: Tom van Dijk





TYPES

Java has two kinds of data types

- Primitive types (int, boolean, double, etc.)
- Reference types (classes and interfaces, for example String, List, etc)

Reference types have a subtyping relation

TYPE THEORY

A little bit of Mathematics: set theory

- Say **T** is a set. For example: **T** = { mammal, human, animal, insect, ant }
- The subtyping relation $\preccurlyeq$ contains all pairs in **T**×**T** such that $(t_1, t_2) \in \preccurlyeq$ if and only if t_1 is a subtype of t_2 (in other words $t_1 \preccurlyeq t_2$ iff t_1 is a subtype of t_2)
  - Example: human$\preccurlyeq$mammal, ant$\preccurlyeq$insect, insect$\preccurlyeq$animal, but not insect$\preccurlyeq$human

TYPE THEORY

The relation $\preccurlyeq$ is a partial order, meaning a binary relation that is:

- reflexive: $t_1 \preccurlyeq t_1$
  Example: human $\preccurlyeq$ human
- transitive: if $t_1 \preccurlyeq t_2$ and $t_2 \preccurlyeq t_3$ then $t_1 \preccurlyeq t_3$
  Example: if ant $\preccurlyeq$ insect and insect $\preccurlyeq$ animal then ant $\preccurlyeq$ animal
- antisymmetric: if $t_1 \preccurlyeq t_2$ and $t_2 \preccurlyeq t_1$ then $t_1 = t_2$
  Example: if x $\preccurlyeq$ human and human $\preccurlyeq$ x, then x = human

TYPE THEORY

T = { mammal, human, animal, insect, ant }

$\preccurlyeq$ = {(mammal,mammal), (human,mammal), (human,human), (animal,animal), (mammal,animal), (human,animal), (insect,animal), (ant,insect), (ant,animal), (ant,ant), (insect,insect)}

TYPE THEORY

Subtyping in programming language theory: substitutability

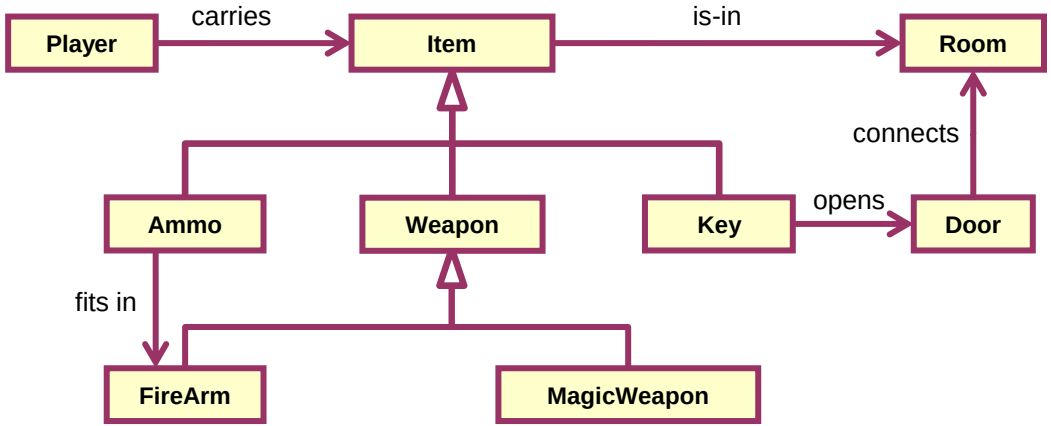
- if S is a subtype of T, then we can safely use S when T is expected
- (Whenever a value of a given type is expected, a subtype can be used)

Example:

- if **Item** is expected, we can use a **Key** because it is-an **Item**
- if **Mammal** is expected, we can use **Dog** but we can't use **Ant**

In Java: S is a subtype of T if S extends T or S implements T

EXAMPLE



- Subtypes of Weapon? FireArm, MagicWeapon, Weapon
- Subtypes of Item? Ammo, Weapon, Key, FireArm, MagicWeapon, Item
- Supertypes of Key? Key, Item, Object

TYPES IN JAVA

- Where does the following program fragment go wrong?

```
Key k1 = new Key();
System.out.println(k1.isPortable());
System.out.println(k1.opens(door));
Item i1 = k1;
System.out.println(i1.isPortable());
System.out.println(i1.opens(door));
Key k2 = i1;
```

`isPortable()` is inherited method of Item
`opens(Door d)` is method of `Key`
subtype value assigned to supertype
`isPortable()` can also be called on `i1`
`opens(Door d)` can *not* be called on `i1`
supertype can *not* be assigned to subtype

- How can this be? i1 and k1 are the same object!

STATIC VERSUS DYNAMIC TYPE OF AN EXPRESSION

- Static type: that which the compiler can infer during “compile-time”
  - Also called declared type
  - Java **will not** infer whether the actual type is a subtype:
    if you declare i1 to be an “Item”, it will be treated as an “Item”, even when it obviously will be a “Key”.
- Dynamic type: that which the value actually has during “run-time”
  - Also called actual type or run-time type
  - i1 has dynamic type Key (because k1 was assigned to it)
  - At some other point, i1 may have dynamic type Item.
- The dynamic type is always a subtype of the static type

STATIC VERSUS DYNAMIC TYPE OF AN EXPRESSION

Java **almost always** uses the static type

- when trying to invoke a method a.b(), it checks whether the declared type of a has a method b
- when assigning a = b, b must be a static subtype of the type of a

But what if you know it's a Key? Can you turn the static type into the actual type?

STATIC TYPE CHANGE (CAST)

Type cast: `(Type) expr` changes the static type to Type

- The only moment when Java uses the dynamic type of an object: to check at runtime if the dynamic type of expr is a subtype of Type

```
Key k1 = new Key();
Item i1 = k1;
System.out.println(i1.isPortable());
System.out.println(((Key) i1).opens(door));
Key k2 = (Key) i1;
```

correct because dynamic type of `i1` is actually `Key` here

Watch the parentheses

- `((Key) i1).opens(door)` is correct: `((Key) i1)` has type "Key".
- `(Key) i1.opens(door)` is wrong: tries to cast the result from the method.

DYNAMIC TYPE TEST

How to find out the dynamic type of an expression expr during “run time”?

- Type test: expr instanceof Type
- This yields true if the dynamic type of expr is a subtype of Type
- null instanceof Type is always “false”.

What does the following print?

```
Key k1 = new Key();
Item i1 = k1;
Item i2 = new FireArm();
Item i3 = null;
System.out.println(k1 instanceof Key);
System.out.println(k1 instanceof Item);
System.out.println(i1 instanceof Key);
System.out.println(i1 instanceof Item);
System.out.println(i2 instanceof Key);
System.out.println(i2 instanceof Item);
System.out.println(i3 instanceof Item);
```

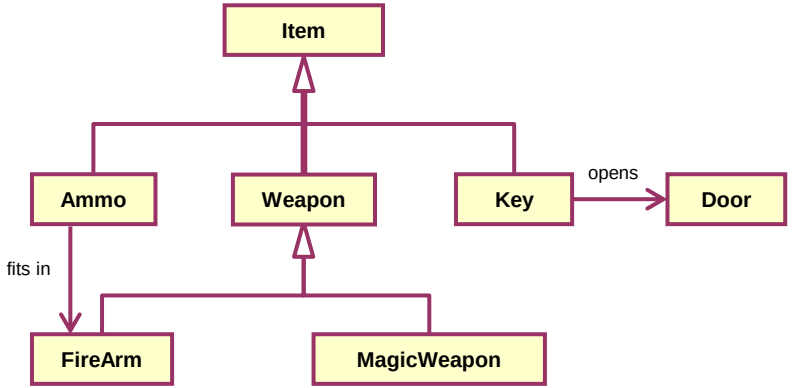
```
true
true
true
true
false
true
false
```

EXAMPLE

```java
public class Player {
    private Item item;

    /** Tests if item is a Key. */
    public boolean hasKey() {
        return item instanceof Key;
    }

    /** Returns the item if it is a key, otherwise null. */
    public Key getKey() {
        return item instanceof Key ? (Key) item : null;
    }

    /** Fires the firearm, if item is a firearm. */
    public boolean fire() {
        return item instanceof FireArm && ((FireArm) item).fire();
    }
}
```

```java
public class FireArm implements Weapon {
    private Ammo ammo;

    public boolean fire() {
        boolean result = ammo != null;
        ammo = null;
        return result;
    }
}
```

TEST




```
Item i1 = new FireArm();
Key k1 = (Key) i1;
Item i2 = null;
Key k2 = (Key) i2;
k2.opens(door);
((FireArm) i1).fire();
FireArm f1 = i1;
FireArm f2 = (Weapon) i1;
FireArm f3 = new Item();
FireArm f4 = (FireArm) new Weapon();
Weapon w1 = (Item) new FireArm();
Weapon w2 = (Weapon) new FireArm();
Weapon w3 = new MagicWeapon();
boolean b1 = w2 instanceof Item;
boolean b2 = w2 instanceof Object;
boolean b3 = w2 instanceof FireArm;
```

Correct: `FireArm` is subtype of Item
Wrong: dynamic type of `i1` is not a subtype of `Key`
Correct: **`null`** is a value of all reference types
Correct: **`null`** is a value of all reference types
Wrong: `k2` is **`null`**
Correct: dynamic type of `i1` is `FireArm`
Wrong: static type of `i1` is Item, not `FireArm`
Wrong: static type of `(Weapon) i1` is `Weapon`
Wrong: static (& dynamic) type of **`new`** `Item()` is `Item`
Wrong: dynamic type of **`new`** `Weapon()` is `Weapon`
Wrong: static type of `(Item)` **`new`** `FireArm()` is `Item`
Correct but cast is unnecessary
Correct: `MagicWeapon` is a subtype of `Weapon`
`b1` becomes **`true`**
`b2` becomes **`true`**
`b3` becomes **`true`**