IT I: Heute

- Klasse *Object*
- *equals*, *hashCode*, *toString*
- *HashSet*

Organisatorisches

- Wissensüberprüfung heute am Papier (um 11:25h), wie immer im Hilbertraum.

Wissensüberprüfung

- Verneinung von *a & b:*
  - *!(a & b)*
  - *!a | !b*
- Verneinung von *x>x' & y>y'*:
  - *!(x>x' & y>y')*
  - *x<=x' | y<=y'*

Wissensüberprüfung

- Aufruf der Methode *makeRandomStep* für Roboter

  - lässt den Roboter zufälligen Schritt machen,

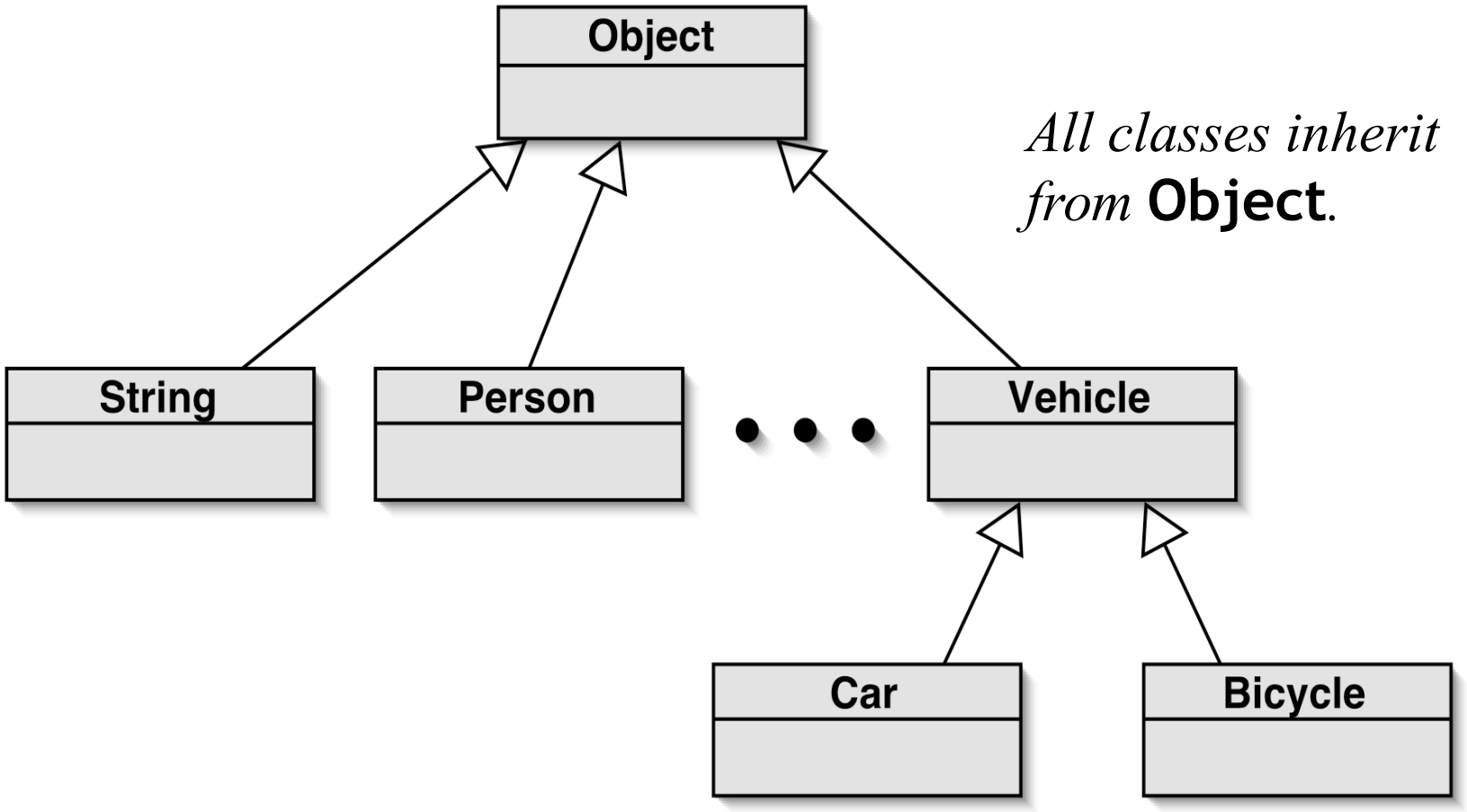
  - gibt Position nach Schritt zurück

- → Mehrfachaufruf lässt Roboter mehrere Schritte machen, d.h. Konstruktionen der folgenden Art funktionieren nicht:

  if(robi.makeRandomStep.getX()>...

  & robi.makeRandomStep.getY()>...)

The *Object* class



All classes inherit from **Object**.

Die Klasse *Object*

- Alle Klassen sind Unterklassen von *Object*.
- *Object* stellt einige Methoden zur Verfügung:
  - String toString()
  - boolean equals(Object obj)
  - int hashCode()
  - Object clone()
  - …

Die Methode *toString()*

- Die Methode *toString()* liefert eine textuelle Repräsentation eines Objekts zurück.
  - Diese soll aber für Objekte unterschiedlicher Klassen verschieden sein, z.B. für String und ArrayList.
  - *toString()* von *Object* liefert den Klassennamen des Objekts und seine Nummer (Hashwert) zurück.
  - Für andere Klassen (die Unterklassen von *Object* sind) kann *toString()* neu definiert werden.

- Wird ein Objekt anstelle eines Strings angegeben, wird automatisch die Methode *toString()* des Objekts aufgerufen.
  - Z.B. in *System.out.println()*

public boolean equals(Object obj)

- Eine weitere Methode, die in *Object* definiert (und implementiert) ist.
- *equals()* soll *true* zurückgeben, wenn das Objekt, für das *equals()* aufgerufen, gleich dem übergebenen Objekt *obj* ist.

public boolean equals(Object obj)

- *equals()* sollte entsprechend die folgenden Eigenschaften haben:
  - *obj.equals(obj) == true*
  - *obj1.equals(obj2) == obj2.equals(obj1)*
  - Wenn *obj1.equals(obj2)==true* und *obj2.equals(obj3)==true* dann auch *obj1.equals(obj3)==true*.
  - Das Ergebnis von *obj1.equals(obj2)* sollte sich nicht verändern, wenn *obj1* und *obj2* unverändert bleiben.
  - *obj.equals(null) == false*

Überschreiben von *equals()*

- *equals(Object obj)* ist für beliebige Objekte als Argumente definiert.
- In einer überschreibenden *equals()*-Methode ist in der Regel ein Überprüfen der Klasse von *obj* und ein Casting notwendig.
- *obj* **instanceof** *eineKlasse*
  - Liefert *true*, wenn *obj* ein Objekt von *eineKlasse* ist.

hashCode()

Um sicherzustellen, dass gleiche Objekte korrekt aufgefunden werden, müssen zwei gleiche Objekte, d.h.

x.equals(y) == true,

auch den gleichen Hashcode haben,

x.hashCode() == y.hashCode().

Die Umkehrung muss/kann i.a. nicht gelten!!!

hashCode()

- Auch für andere Bibliotheksklassen muss *hashCode()* korrekt implementiert sein.
- Überschreiben von *equals()* erfordert das Überschreiben von *hashCode()*!
- Eine neue *hashCode()*-Methode verwendet in der Regel die Attribute, die von *equals()* verwendet werden.
  - z.B.: Kombination der Hashcodes der Attribute mit XOR(^):
    a1.hashCode() ^ a2.hashCode() ^ a3.hashCode()
    für Attribute a1, a2, a3.

hashCode()

Generell gilt:

- Werden in *equals()* Attribute *a1*,*a2*,... verwendet, so ist jede Funktion, die die Attributwerte (bzw. ihren Hashcode) auf einen *int*-Wert abbildet, ein gültiger Hashcode.

hashCode()

Vereinfachte Illustration für interne Verwendung des Hashcodes:

- Speichere Objekte einer Klasse in einem Array auf dem Index, der dem Hashcode entspricht.
- Auf diese Weise hat man schnellen Zugriff auf Objekt (weil über Hashcode direkt zugreifbar).

- Probleme gibt's, wenn zwei verschiedene Objekte denselben Hashcode haben.

hashCode()

Generell gilt:

- Java verwendet *hashCode()* intern zum Speichern von Objekten.
- Ein expliziter Aufruf von *hashCode()* ist nicht sinnvoll!

HashSet

- HashSet verfügt über Methoden (ähnlich wie ArrayList):
  - *add()*
  - *size()*
  - *for each* Schleife

Achtung:
Für *HashSet* gibt's keine *get*-Methode!
(Zugriff nur über *for each* Schleife!)

HashSet

- Unterschiede zwischen *ArrayList* und *HashSet*:
  - *ArrayList* enthält eine Sequenz von Objekten.
  - Objekte können öfters vorkommen.
  - *HashSet* enthält Objekte in keiner bestimmten Reihenfolge (keine *get*-Methode!)
  - In *HashSet* kommt ein Objekt nur einmal vor.

HashSet

- Benötigt eine korrekte Implementierung der Methoden

  public int hashCode()
  public boolean equals(Object ob)

  in *java.lang.Object*.

- Die Methode *hashCode()* gibt im wesentlichen an, wo ein Objekt in einem *HashSet* gespeichert wird.