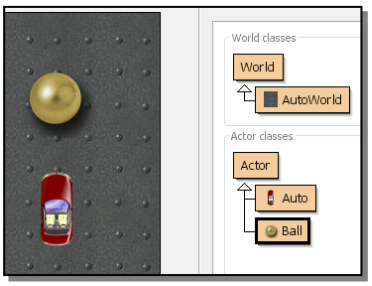
OOP: AUTO

Unser Ziel ist es, ein Spiel zu erstellen, das so ablaufen soll: Der Spieler steuert ein Auto, das sich von unten nach oben bewegt. Sobald das Auto oben angekommen ist, wird es nach unten „teleportiert“ und fährt wieder von unten nach oben. Dabei darf das Auto nicht mit den Kugeln zusammenstoßen, die sich automatisch über das Spielfeld bewegen, sonst wird das Auto zerstört und der Spieler hat das Spiel verloren.



Klassen AutoWorld, Auto, Ball / Welt füllen / Auto bewegen

(→ 02_Auto)

Wir erstellen eine Welt (Klasse: AutoWorld*) mit 1000px x 800px. Bild: backgrounds / rivets.jpg*

```
public class AutoWorld extends World
{
   public AutoWorld()
   {
      super(1000, 800, 1);
   }
…..
```

Wir erstellen eine Klasse Auto *(Bild: transport / car02-n.png) und eine Klasse* Ball *(Bild: objects / gold-ball.png). Die Objekte der Klasse* Ball *„können“ bisher nichts, hier muss also noch kein Code geschrieben werden. Die Objekte der Klasse* Auto *sollen sich automatisch von unten nach oben bewegen, sobald das Spiel startet.*

```
public class Auto extends Actor
{
  public void act()
   {
      // Auto fährt kontinuierlich nach oben:
      this.moveUp();

      // Tastatursteuerung links und rechts:
      if(Greenfoot.isKeyDown("right"))
      {
         this.moveRight();
      }

       if(Greenfoot.isKeyDown("left"))
      {
         this.moveLeft();
      }
   }

   public void moveUp()
   {
      this.setLocation(getX(), getY() - 5);
   }

   public void moveLeft()
   {
     setLocation(this.getX()-5, this.getY());
   }
```

```
    public void moveRight()
    {
      this.setLocation(this.getX()+5, this.getY());
    }
}
```

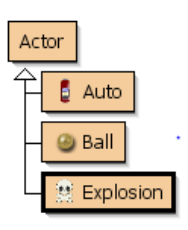
Als letztes müssen wir noch die Welt mit dem Auto und einem Ball füllen:

```
public class AutoWorld extends World
{
    public AutoWorld()
    {
        super(1000, 800, 1);
        this.fillWorld();
    }

    public void fillWorld()
    {
        Auto meinAuto;
        meinAuto = new Auto();

        // this.addObjekt(objekt, x, y);
        this.addObject(meinAuto, 500, 750);

        Ball ball1;
        ball1 = new Ball();
        this.addObject(ball1, 500, 200);
    }
}
```

Kollision Auto / Ball

(→ 03_Auto)

Wenn das Auto mit einem Objekt der Klasse Ball *zusammenstößt, soll das Auto vernichtet werden. Hierzu erstellen wir die Klasse* Explosion *(Bild: symbols / skull.png). In dieser Klasse brauchen wir keinen Code. Bei einem Zusammenstoß soll einfach ein Objekt der Klasse Explosion erscheinen und das Auto-Objekt gelöscht werden.*




```
public class Auto extends Actor
{
  public void act()
    {
        ….

        this.checkCollision();
    }

public void checkCollision()
    {
        // Kollision mit Ball:
        Actor meinBall;
        meinBall = this.getOneIntersectingObject(Ball.class);

        if(meinBall != null)
        {
          // Wir erstellen Variablen, um uns die Position des Autos zum Zeitpunkt der Kollision zu merken
          int x;
          int y;

          // Koordinaten des Autos einlesen:
```

```
        x = this.getX();
        y = this.getY();

        Explosion ex;
        ex = new Explosion();
        this.getWorld().addObject(ex, x, y);

        this.getWorld().removeObject(this);
      }
}
```

Klassen AutoWorld, Auto, Ball / Welt füllen / Auto bewegen

(→ 04_Auto)

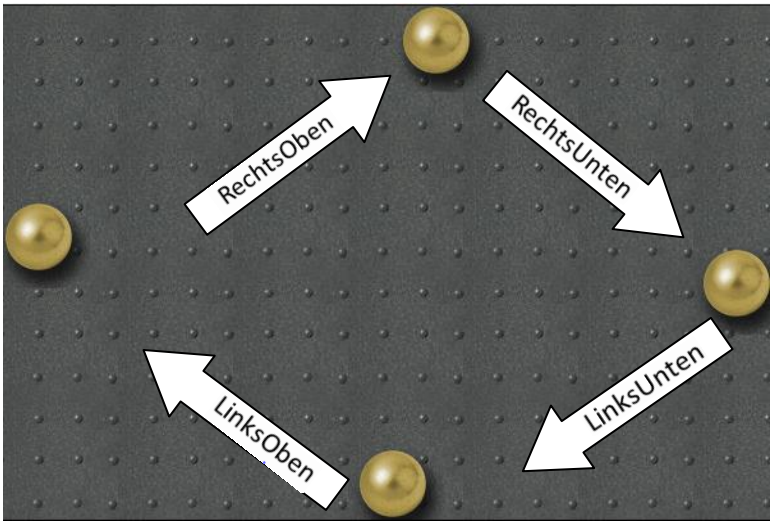
Sobald das Auto-Objekt oben angekommen ist, soll es auf der gleichen X-Koordinate nach unten „teleportiert“ werden.

```
public class Auto extends Actor
{
  public void act()
   {
      // Auto fährt kontinuierlich nach oben:
      this.moveUp();

      // Prüfen, ob Auto oben angekommen ist:
      // Dann wird Auto wieder nach unten gesetzt
      if(this.getY() < 50)
      {
         this.setLocation(this.getX(), 800);
      }

      ……
}
```

Klassen AutoWorld, Auto, Ball / Welt füllen / Auto bewegen

(→ 05_Auto_SWITCH)

Wenn das Ball-Objekt an den Rand des Spielfelds stößt, soll das Objekt abprallen. Damit das Ball-Objekt physikalisch korrekt seine Richtung ändert, muss man sich die aktuelle Richtung des Objekts in einem Attribut merken.
Beispiel: Wenn der Ball am Rand anstößt und die Richtung RechtsOben hatte, soll sich das Objekt ab jetzt nach RechtsUnten bewegen.



In Worten ausgedrückt soll das Programm so aussehen:

WENN Ball-Objekt an den Rand des Spielfelds stößt, DANN ändere *meineRichtung* so:

FALLS meineRichtung = „RechtsOben“ ➔ meineRichtung = „RechtsUnten“
FALLS meineRichtung = „RechtsUnten“ ➔ meineRichtung = „LinksUnten“
FALLS meineRichtung = „LinksUnten“ ➔ meineRichtung = „LinksOben“
FALLS meineRichtung = „LinksOben“ ➔ meineRichtung = „RechtsOben“

Ball-Objekt stößt am Rand des Spielfelds an

Man erkennt, dass das Ball-Objekt am Rand anstößt, wenn:

this.getY() < 20 ODER this.getY() > 780 ODER this.getX() < 20 ODER this.getX() > 980

ODER drückt man in JAVA mit zwei senkrechten Strichen aus, also so: ||
Dieses Zeichen erhält man mit *AltGr und* <

Unser korrekter Code sieht so aus:

```
public class Ball extends Actor
{
   // Wir merken uns die Richtung des Balls im Attribut meineRichtung;
   private String meineRichtung = "RechtsOben";

   public void act()
   {
      // Wenn das Ball-Objekt an den Rand des Spielfelds stößt, soll es die Richtung ändern:
      if(this.getY() < 20  || this.getY() > 780  || this.getX() < 20 || this.getX() > 980)
      {
        switch(meineRichtung)
        {
          case "RechtsOben": meineRichtung = "RechtsUnten";
                             break;
          case "RechtsUnten": meineRichtung = "LinksUnten";
                              break;
          case "LinksUnten": meineRichtung = "LinksOben";
                             break;
          case "LinksOben": meineRichtung = "RechtsOben";
                            break;
        }
      }
```

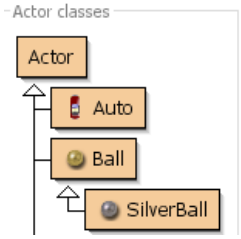
switch() *und* ***if()*** *funktionieren ganz ähnlich. Man kann mit beiden verschiedene Fälle unterscheiden und sagen, was in welchem Fall passieren soll. Mit* ***if()*** *wird es aber sehr schnell unübersichtlich, wenn man mehr als zwei Fälle hat. Dann ist switch() hervorragend geeignet!*

if() nennt man übrigens „Alternative“
switch() nennt man „Fallauswahl“

Den break-Befehl braucht man, damit die Fallauswahl abgebrochen wird, sobald man den richtigen Fall gefunden hat. Falls Ihnen das nicht einleuchtet: Probieren Sie mal aus, was passiert, wenn Sie die break-Befehle löschen!

Vererbung: SilverBall

(→ 06_Auto)



Zusätzlich soll nun zeitgesteuert alle 300 Millisekunden ein neuer Ball im Spielfeld erscheinen, der sich von links nach rechts bewegt. Weil sich diese neuen Ball-Objekte aber anders bewegen sollen als der bisherige Ball, brauchen wir hierfür eine neue Klasse, die wir *SilverBall* nennen!

Da diese neuen Ball-Objekte aber eigentlich die gleichen Methoden wie unser goldener Ball haben sollen (moveUp, moveLeft…) wenden wir hier das Konzept der **Vererbung** an! Die Klasse *SilverBall* soll alle Attribute und Methoden von der Klasse *Ball* erben. Das erreichen wir mit Rechtsklick auf die Klasse Ball (also NICHT auf Actor!) / New Subclass…

Dass die neue Klasse SilverBall alle Attribute und Methoden der Klasse Ball erbt, erkennt man im JAVA-Programmcode daran, dass am Beginn der Klasse *extends Ball* steht. Der komplette Programmcode dieser Klasse sieht so aus:

```
public class SilverBall extends Ball
{

   public void act()
   {
      // Die Methode moveRight erbt diese Klasse von der Klasse Ball!
      this.moveRight();
   }
}
```

Ihnen müsste auffallen, dass wir die Methode moveRight*() aufrufen können, obwohl wir diese Methode gar nicht in der Klasse SiverBall programmiert haben. Hier können Sie sehen, wie Vererbung funktioniert. Die* Subclass *(hier: SilverBall) erbt diese Methode von der* Superclass *(hier: Ball).*

Damit jetzt noch alle 300 Millisekunden ein neuer SilverBall eingefügt wird, ergänzen wir die Klasse AutoWorld so:

```
public class AutoWorld extends World
{
   // Attribut zaehler: Wir wollen mitzählen, wie oft die act()-Methode aufgerufen wurde,
   // weil alle 300ms ein neuer SilverBall erscheinen soll
   private int zaehler = 0;

   public AutoWorld()
   {
      // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
      super(1000, 800, 1);
      this.fillWorld();
   }

   public void act()
   {
      // Alle 300 Millisekunden einen neuen Ball einfügen:
      zaehler = zaehler + 1;

      if(zaehler == 300)
      {
        this.CreateBall();
        zaehler = 0;
      }
   }
```

```
   public void CreateBall()
   {
      SilverBall meinBall;
      meinBall = new SilverBall();
      this.addObject(meinBall, 0, Greenfoot.getRandomNumber(400));
   }
}
```

Ihnen müsste noch eine andere erstaunliche Sache auffallen: Wenn das Auto mit einem SilverBall zusammenstößt, kommt es genauso zu einer Kollision (und das Spiel endet) wie bei einem Zusammenstoß mit dem goldenen Ball.

Warum ist das so? In der Auto-Klasse fragen wir doch nur ab, ob das Auto mit einem Ball zusammenstößt:

meinBall = getOneIntersectingObject(**Ball.class**);

Ein SilverBall taucht hier im Programmcode überhaupt nicht auf. Trotzdem funktioniert es. Grund: Ein SilverBall ist auch ein Ball, weil der SilverBall alles vom Ball erbt. Man könnte auch sagen: Ein Objekt der Klasse SilverBall ist eine Sonderform der Klasse Ball.

SilverBall prallt ab
(→ 07_Auto)

Die Objekte der Klasse Silverball rollen bisher von links nach rechts übers Spielfeld und bleiben dann an der rechten Begrenzung „kleben“. Wir sorgen jetzt dafür, dass diese Objekte „abprallen“, also an der Begrenzung ihre Richtung ändern.

```
public class SilverBall extends Ball
{
   private String meineRichtung = "Rechts";

   public void act()
   {
      // Ball-Objekt muss die Richtung ändern, wenn: x > 980 und sich das Objekt nach rechts bewegt!
      // Wenn man die aktuelle Richtung nicht beachten würde, dann würden auch Bälle, die gerade
      // abgeprallt sind, wieder ihre Richtung ändern (weil bei denen auch x > 980 ist!)
      if(this.getX() > 980)
      {
         if(meineRichtung == "Rechts")
         {
            meineRichtung = "Links";
         }
      }

      // Ball-Objekt muss die Richtung ändern, wenn: x < 20 und sich das Objekt nach links bewegt!
      if(this.getX() < 20)
      {
         if(meineRichtung == "Links")
         {
            meineRichtung = "Rechts";
         }
      }

      if(meineRichtung == "Rechts")
      {
         this.moveRight();
      }

      if(meineRichtung == "Links")
      {
         this.moveLeft();
      }
   }
}
```

Anzahl der SilverBalls begrenzen
(→ 08_Auto)

```
public class AutoWorld extends World
{
……

   // Wir zählen die Anzahl der erzeugten Silverballs mit:
   private int anzahlSilverballs = 0;

…..

   public void act()
   {
      // Alle 300 Millisekunden einen neuen Ball einfügen:
      zaehler = zaehler + 1;

      // Neue Silerballs sollen nur erzeugt werden, wenn seit dem letzten
      // Objekt 300ms vergangen sind UND es weniger als 5 Silverballs gibt.
      // Und drückt man in JAVA so aus: &&
      if(zaehler == 300 && anzahlSilverballs < 5)
      {
        this.CreateBall();
        zaehler = 0;
        anzahlSilverballs = anzahlSilverballs +1;
      }
   }
}
```

Auto kann nach unten fahren / Teleportation nach oben
(→ 09_Auto)

```
public class Auto extends Actor
{
  // Wir merken uns unsere Fahrtrichtung:
  private String meineRichtung = "Oben";

  public void act()
   {
      // Falls Pfeiltaste (hoch / runter) gedrückt ist: Richtung merken!
      if(Greenfoot.isKeyDown("down"))
      {
         meineRichtung = "Unten";
      }

      if(Greenfoot.isKeyDown("up"))
      {
         meineRichtung = "Oben";
      }

      // Tastatursteuerung hoch / runter:
      if(meineRichtung == "Oben")
      {
        this.moveUp();
      }

      if(meineRichtung == "Unten")
      {
        this.moveDown();
      }

      // Prüfen, ob Auto OBEN angekommen ist:
      // Dann wird Auto wieder nach UNTEN gesetzt
      if(this.getY() < 50)
      {
         this.setLocation(this.getX(), 750);
      }
```

```
        // Prüfen, ob Auto UNTEN angekommen ist:
        // Dann wird Auto wieder nach OBEN gesetzt
        if(this.getY() > 750)
        {
           this.setLocation(this.getX(), 50);
        }
        …

        this.checkCollision();
    }

    public void moveUp()
    {
        this.setLocation(getX(), getY() - 5);
    }

    public void moveDown()
    {
        this.setLocation(getX(), getY() + 5);
    }

    …
}
```