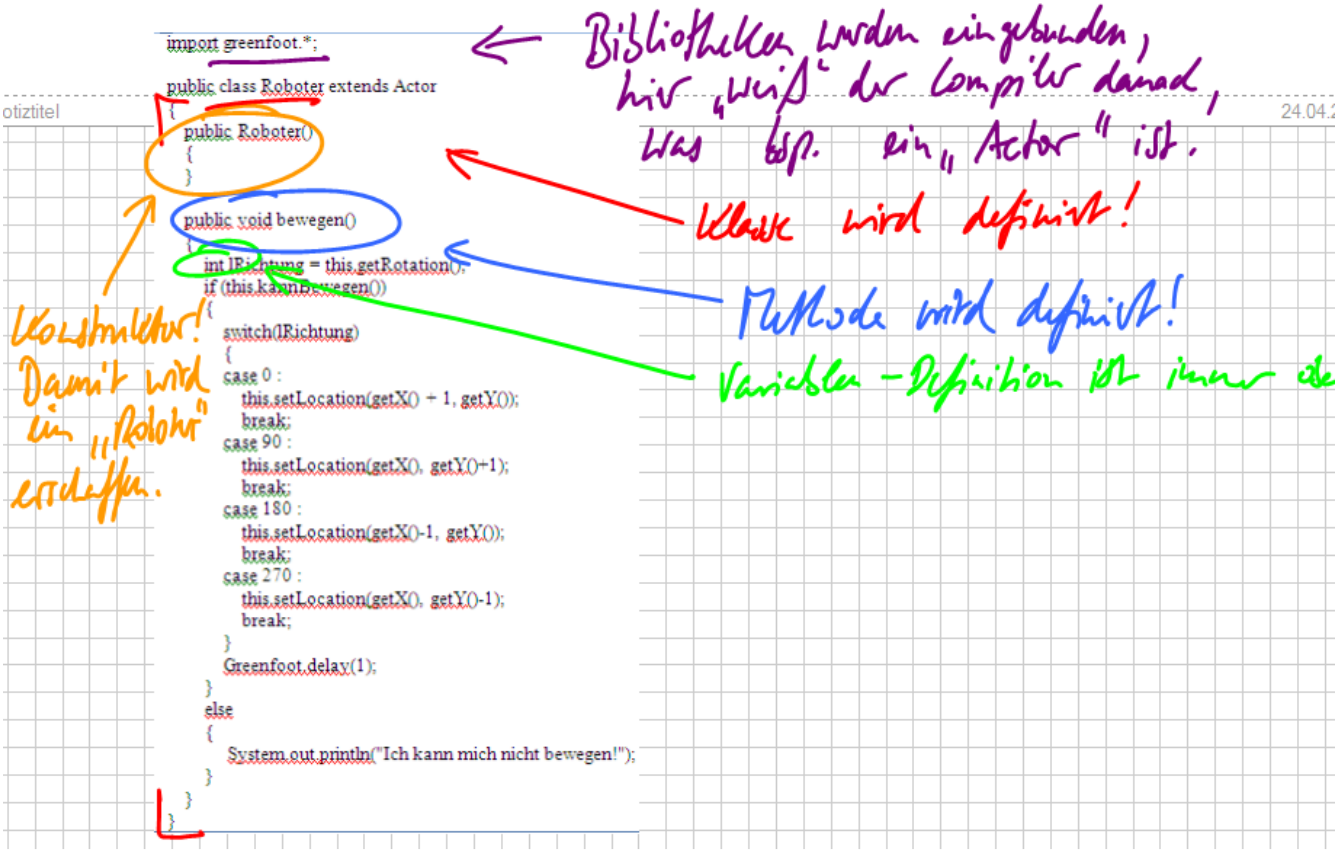
EI Info 2011-12	*INFORMATIK* 2. Klausur (24.04.2012)	

1. Aufgabe **(3 Punkte)**

Nenne die wesentlichen Teile eines Java-Programmes anhand eines Ausschnittes des Quellcodes der Klasse „Roboter" in Greenfoot:



2. Aufgabe **(2 Punkte)**

Robby, Robita und Robson sind Unterklassen der Klasse Roboter. Erläutere anhand dieser Tatsache die Vorteile der objektorientierten Programmierung.

Die drei Unterklassen „erben" die Methoden der Klasse Roboter (falls diese public sind), was eine Erleichterung für den Programmierer bedeutet („Vererbung").

Ein komplexes Programm lässt sich in überschaubare Einheiten zergliedern. Insbesondere, wenn mehrere Programmierer unabhängig voneinander arbeiten, ist dies vorteilhaft („Kapselung").

3. Aufgabe **(2 Punkte)**

Wann hebt Robby bei dieser Bedingung den Akku auf? Begründe deine Antwort kurz!

```
if (!akkuAufFeld() && (wandVorne() || (wandLinks() && !wandRechts()) ) )

{          akkuAufnehmen();          }
```

Nie. Denn die if-Bedingung wird nur erfüllt, wenn kein Akku dort ist, was nie geht (wegen nicht-akkuAufFeld und dem folgenden &&).

4. Aufgabe **(13 Punkte)**

Robita kennt folgende act-Methode:

```
public void act()
{
        if (wandVorne() && akkuAufFeld())
        {
          System.out.println("Warum?!");

                    if (akkuAufFeld())
          {
                    akkuAufnehmen();
                              bewegen();
          }
        }

        else
    {
                              bewegen();
        }
}
```

a) Beschreibe schrittweise, welcher Vorgang abläuft, wenn du bei dem folgenden Aufbau die Play-Taste drückst:

Die Bedingung wandVorne UND akkuAufFeld ist nur im 4. Schritt erfüllt. Daher bewegt sich Robita erst einmal über die ersten zwei Akkus und das freie Feld hinweg. Im vierten Durchlauf steht Robita vor der Wand UND der Akku liegt auf dem Feld. Also wird sie „Warum?!" sagen und sich nicht mehr bewegen.

b) Lass Robita anstelle von „System.out.println("Warum?!");" alle ungeraden Zahlen von 1 bis 99 aufsagen! Welche Struktur und welche Vorteile hat dabei eine for-Schleife?

Robita bekommt diese Anweisung:

```
for (int i=0; i<50; i++)
{
      System.out.println(2*i+1);
}
```

Die Vorteile einer for-Schleife liegen auf der Hand; es ist weit einfacher, dem Rechner ein „Kochrezept" zu geben, was er dann immer wieder ausführt. Ansonsten müsste man alles selbst in den Quellcode eintippen!

c) Lass Robita nun nur die Primzahlen im Zahlbereich von 1 bis 100 aufsagen. Verwende dazu eine Methode „public boolean primzahlTest(int i)" mit der du bei jeder Zahl testen kannst, ob es sich um eine Primzahl handelt. Gib den Quelltext der Methode ebenfalls an.

Für die Primzahlen lässt man das Println via eine if-Abfrage raus:

if (primzahlTest(i)) {println(i);}, wobei das i das i aus der Vorschleife ist.

Die Methode selbst muss man sich erst einmal überlegen und das ist nicht so einfach. Eine naive Lösung ist, die Testzahl i nacheinander durch alle Vorgängerzahlen k zu teilen und zu prüfen, ob das Ergebnis eine ganze Zahl ist. Dann wäre i KEINE Primzahl. Dazu braucht man eine Methode, die prüft, ob eine Zahl eine ganze Zahl ist. Dies schafft man wiederum, wenn man immer wieder (maximal i-mal, dann stoppt die Schleife!) 1 vom Ergebnis abzieht und schaut, ob mal 0 herauskommt. Ist dem so, hat man eine ganze Zahl und damit ist i KEINE Primzahl.

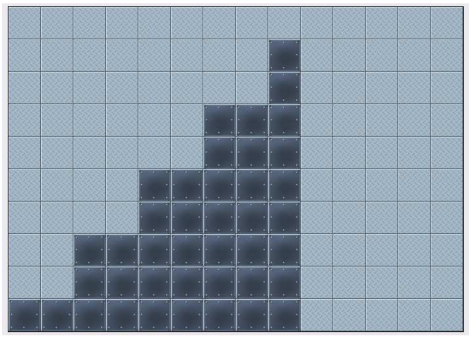
Was hier aber auch geht, ist, von i immer wieder die Zahl k abzuziehen. Sollte i ein Vielfaches von k sein, stößt man zwangsläufig auf Null. Damit wäre i keine Primzahl. Notieren wir das:

```
for (int i=1; i<101; i++)
{
	int test;
	boolean prim=true;

	for (int k=2; k<i; k++)
	{
		test = i;

		for (int j=0; j<i; j++)
		{
			test = test – k;
			if (test==0) {prim=false;}
		}
	}

	if (prim) {System.out.println(i);}
}
```

Man kann diese Lösung noch sehr optimieren, aber das ist zum Lösen dieser Aufgabe nicht nötig.

5. Aufgabe **(4 Punkte)**

Schreibe eine Methode „treppenLauf()" für den sportlichen Roboter Robben, damit er Treppen dieser Art (die Anzahl der Stufen zur Spitze, hier 3, sind nicht bekannt) erklimmen kann:



Dabei soll er allerdings immer bei der zweitobersten Treppenstufe zum Stehen kommen. Verwende hierzu die Methoden „bewegen()", „dreheRechts()" und „wandVorne()".

Für dieses Problem sollte man sich einige weitere Methoden aus den obigen drei definieren. Bspw. dreheLinks als 3xdreheRechts und die Wandtester für alle Seiten.

Dann brauchte es eine Methode „erklimmeTreppe", mit der eine komplette Stufe abgelaufen wird. Diese wird ausgeführt, wenn wandVorne true ist. Sie lautet dreheLinks, 2xbewegen, dreheRechts, 2xbewegen.

Diese kann man jede Runde ausführen via die act-Methode. Allerdings kommt irgendwann der Fall, dass vor Robby keine Wand mehr ist UND keine mehr unten. In diesem Fall ist er nicht mehr auf der Treppe, sondern über die Spitze hinweggegangen. Nun muss man die Methode laufeZurueck ausführen, die ihn auf die vorletzte Stufe bringt. Sie entspricht einem umgekehrten erklimmeTreppe.

Zusatz ***+1 Punkt***

Schreibe eine Methode für Robben, um dem Roboter Robery, der auf der dritten Stufe steht, auszuweichen. Dazu darf Robben über ihn klettern. [sonst Robery.AugeBlau(Robben)].

Mit einer Prüffunktion roberyVorne schaut man, ob Robery auf dem Feld steht. In diesem Fall führt man ein dreheLinks, bewegen, dreheRechts, bewegen, bewegen, dreheRechts, bewegen, dreheLinks. Das ist es.