

SCOOP example projects:

1) Santa Claus

Santa lives in his little house in Jyëvväskÿølgbrø; his favourite activity is to have a nap. So he sleeps all the time, except for rare moments when he is awoken by either a group of elves or a group of reindeer. There are 10 elves who live in the area and work in a toy workshop. They produce toys and, once in a while, they run out of ideas and need to ask Santa for help. They can only wake up Santa if they form a group of three. There are also nine reindeer that live in the stable nearby; Santa uses them to deliver toys to kids. The reindeer are as lazy as Santa himself, so for the most of the day they just sleep. When they want to work, they need to form a group of nine (that is all the reindeer must join) and wake up Santa. If they manage to do it, Santa does the delivery round and then goes back to sleep (so do the reindeer). It is important to note that in a situation when there is a group of three elves and the group of nine reindeer trying to wake up Santa at the same time, it is the reindeer that get the priority.

Your application should be based on classes SANTA, REINDEER, ELF, plus any additional classes that implement useful abstractions. Use inheritance whenever necessary – remember that abstraction is beautiful and there is nothing uglier than an application with several classes that look almost the same and contain identical code.

2) Active objects with Santa Claus

The goal of this task is to find an elegant and efficient way to simulate active objects in SCOOP. SCOOP does not directly support the concept of active object but we should be able to implement scenarios that require the use of rendezvous synchronisation. Our beloved Santa has agreed to serve as example once again.

Santa has finally decided to go with the wind of change and make his business Internet-based. Before going online, he wants to try out locally a simple client-server approach to optimise his consultancy activities. From now on, the elves who want to consult Santa are not required to form a group anymore – they may simply send an individual request to Santa's server and wait until Santa accepts to meet them using a new teleconferencing support. After issuing a request, an elf is blocked until Santa has accepted his request and met him. After the consultation the elf should work on his own for a while, then play with his kids (each elf has three kids; the kids are always ready to play with their dad, except when they are eating). Important: elves never sleep, they are always active!

Unlike elves, Santa sleeps most of the time. Nevertheless, he wakes up once in a while to check if there are any requests from the elves. If there are any pending requests, he accepts a request and consults the elf who issued the request. Santa does not need to process all pending requests – in fact, we should be able to parametrise Santa to accept at most n requests in a row. After satisfying a given number of requests Santa goes back to sleep. A very important thing: the old Santa is able to process just one request (that is meet just one elf) at a time!

Oh, one more thing: Santa has optimised the business and he only does consulting; UPS has taken over his delivery business, so no reindeer are involved anymore. They were useless anyway, weren't they?

Your task is to implement the scenario in SCOOP. Obviously, the client-server scenario described above calls for a rendezvous-style synchronisation. You need to implement active objects that represent Santa, elves, and elves' kids. Make sure that their schedules (bodies) specify correctly their activities:

- SANTA: sleep, accept some pending request, process the request (i.e. meet the elf), and so on. Remember that accepting a request is a blocking activity.
- ELF: issue a request to Santa, work a bit on your own, play with your kids. Remember that after issuing a request the elf is blocked until Santa accepts the request. Also, the elf has to play with all his kids (not necessarily at the same time) before issuing the next request to Santa.
- KID: eat, play with your dad, eat, play with your dad, and so on.

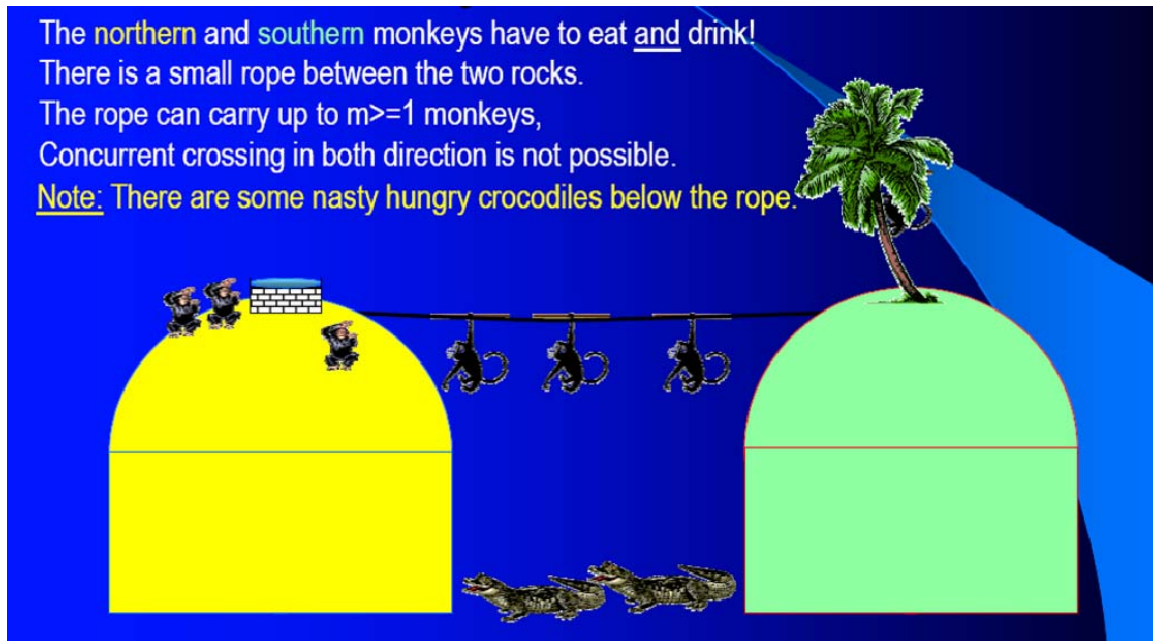
Once again, make sure that you make an appropriate use of inheritance to achieve the necessary level of abstraction. Try to provide a general class (or set of classes) that implements an active object and then derive specialised classes for Santa, elves, and kids. Try to parametrise Santa in such a way that he accepts at most n pending requests before going back to sleep.

Sleeping, elves' own work, eating, and playing can be represented with simple routines nap, work, eat, and play. The details are irrelevant – it is enough to output a corresponding message, e.g. “Santa is sleeping”.

3) Monkeys

The northern and southern monkeys have to drink **and** eat! There is a small rope between the two rocks. The rope can carry up to $m \geq 1$ monkeys, concurrent crossing in both direction is not possible.

Note: there are some nasty hungry crocodiles below the rope.



4) The Sleeping Barber

The shop has a barber, a barber chair, and a waiting room with several chairs. When a barber finishes cutting a customer's hair, the barber fetches another customer from the waiting room if there is a customer, or stands by the barber chair and daydreams if the waiting room is empty. A customer who needs a haircut enters the waiting room. If the waiting room is full, the customer comes back later. If the barber is busy but there is a waiting room chair available, the customer takes a seat. If the waiting room is empty and the barber is daydreaming, the customer sits in the barber chair and wakes the barber up.

5) Car Cruise Control

Simulates the behaviour of a cruise control system for a car. After starting the engine, to accelerate, keep pressing the accelerate button. To brake, keep pressing the brake button.

See http://www.doc.ic.ac.uk/~jnm/book/book_applets/CruiseControl.html and http://www.doc.ic.ac.uk/~jnm/book/book_applets/FixedCruiseControl.html

6) Ornamental Garden Problem

A large ornamental garden is open to members of the public who can enter through either one of two turnstiles to the East and to the West of the garden. The management want to determine how many people are in the garden at any one time. They design a computer system to do this. In the program produced, each turnstile is represented by a thread and updates a shared Counter object.

See http://www.doc.ic.ac.uk/~jnm/book/book_applets/Garden.html

7) Readers and Writers

The program allows Readers concurrent access to a resource while restricting Writers to exclusive access. Thread access to the shared resource is depicted by a light blue (cyan) arc segment. It is possible by starting the Reader threads at different times to create an execution in which Writer threads never get access to the resource.

See http://www.doc.ic.ac.uk/~jnm/book/book_applets/ReadersWriters.html

and

Readers and Writers

This version gives priority to Writers by making Readers defer to waiting Reader threads. Consequently, Writers do not starve, however, giving Writers priority means that Readers may starve.

See http://www.doc.ic.ac.uk/~jnm/book/book_applets/ReadWritePriority.html

8) Golf Ball Allocation Monitor

At a golf club, players can hire golf balls for their game from the club and return them to the club after use. The better players, who tend not to lose any balls, only hire one or two. The less experienced players hire more balls, so that they will have spares during the game in case of loss. They are, however, required to buy replacements for the lost balls so that they return the same number that they originally hired. The golf balls are kept by the club groundsman, who turns out to be a techie. He decides to treat the players as Java threads and to write a monitor to allocate golfballs to players, if available, or to delay the players if insufficient are available.

See http://www.doc.ic.ac.uk/~jnm/book/book_applets/GolfClub.html

and

Fair Allocation Strategy

At first things go well, but then he finds that he is constantly getting complaints from the novice players. In some circumstances, novice players might wait forever because they are continually overtaken by better players who require fewer balls. He decides on a policy of allocating strictly in arrival order.

See http://www.doc.ic.ac.uk/~jnm/book/book_applets/GolfClubFifo.html

9) Supervisor-Worker

The program computes the area under a curve using a set of four worker threads.

See http://www.doc.ic.ac.uk/~jnm/book/book_applets/SupervisorWorker.html

10) Parcel - Router

Click on one of the colored buttons to generate a new parcel. The slider at the side of the display adjusts the speed at which parcels drop through the sorter. Generate a stream of alternately colored parcels to see what happens.

See http://www.doc.ic.ac.uk/~jnm/book/book_applets/ParcelSorter.html