Machine Knitting = Design with Constraints

Administrivia

- HW4 + Fabrication Artifact
  - resubmission deadline: April 26
- HW5
  - deadline: April 23
  - resubmission deadline: May 17
- Final project
  - work session: April 23
  - presentation: April 28
  - report + deliverables: May 15

Agenda Today

- View (machine) knitting through the lens of constraints
  - What kinds of constraints?
  - When to check?
- Extending the `KnitoutWriter`
  - `KnittingHelper` and `ShapingHelper` (with shaping helpers for those who hope to work with shaping in their projects)

What do we mean by “Constraint”?

- a rule
- defining the limits of things
- any examples (from everyday life)?
  - two classes cannot be scheduled in the same room at the same time
  - this computer has 1TB storage max
  - the assignment is due in 2 days
  - and of course, knitting! or any activity that is (physically) creating things

A slightly philosophical thought...

- Constraints do not oppose creativity
  - They limit the space of possibilities so that problems become solvable and designs physically realizable; each problem with different constraints makes the problem itself and therefore the solution unique
  - Being able to name the constraints also points to directions that might lift the constraints
  - Exploring within constraints is no less creative than exploring without (think poetry, music)

some reference: https://www.hermanmiller.com/stories/why-magazine/design-q-and-a-charles-and-ray-eames/

Back to Knitting Constraints

- Poll time! Any knitting constraints you can think of? We have seen several types in the course so far.
- In these constraints, what are ones that would *make the knitting fail or the machine stop working*?
  - These are called **hard constraints**, and the rest are **soft constraints** (like guidelines and rules)
- Let's go over these constraints and how we should reason about them (in pattern design, knitout programming, or during fabrication)

Fundamental Knitting Constraint

- loops have to be pulled through existing loops (except for cast-ons that are special)
- hard constraints because violating this will cause:
  - no knitting performed
  - failed knitting, disconnected pieces

Fundamental Knitting Constraint Handling

- In pattern programming:
  - make sure in your pattern you do not just suddenly knit a row with additional loops without properly increasing from the previous row

Machine Constraint

- the knitting machine poses specific constraints
  - the max racking value
  - the max # yarn carriers
  - the needle gauge
  - stacked loops on a needle cannot separate
  - front bed only knits, back bed only purls
  - only 1 yarn inhooking device so you cannot inhook yarns without releasehook before inhooking the next one, and outhook has to be performed for any yarn that's inhooked
  - loops transferred have to be done head-on
  - ... many more like the possible loop sizes, yarn thicknesses, both hard and soft constraints

Machine Constraint Handling

- Not satisfying the machine constraints is going to produce unknittable patterns or faulty patterns!
- In pattern programming:
  - encode all these constraints as checks in the code if possible, like how the `KnitoutWriter` class in HW5 works

Yarn Tension Constraint

- how tight the yarn is when feeding through the machine
- hard-ish constraint
  - too loose: yarn pile up on the machine, potentially dropping loops
  - too tight: loops drop from needles when transferring, yarn breaks
  - it causes faulty knits but does not prevent the machine from running

Yarn Tension Constraint Handling

- In pattern programming:
  - avoid very large racking values
  - if you need to increase multiple stitches, consider spacing them out; do not try to do them in a single pass to avoid adding too much tension to the yarn
  - adding waste regions to stabilize yarn before entering pattern region
- In fabrication:
  - check yarn tension before fabricating, pulling onto the thread and check the yarn passes through smoothly everywhere (if you feel resistance it might be tangled somewhere; if too loose, use the top tensioning device to add tension)

Float Length Constraint

- Avoid long floating yarn in the back of the pattern
- Soft constraint, mostly for aesthetics
  - This really happens when you use multiple carriers in the same row (so colorwork)

Float Length Constraint Handling

- In pattern programming:
  - use the strategy of double-bed jacquard knitting to also use the floating yarn (unused on the front bed) on the back bed, or knit with both if there are only two yarns through plating
  - or think of how you would use more than one carriers with the same color to avoid floats (doing intarsia basically)

Machine Programming Constraint

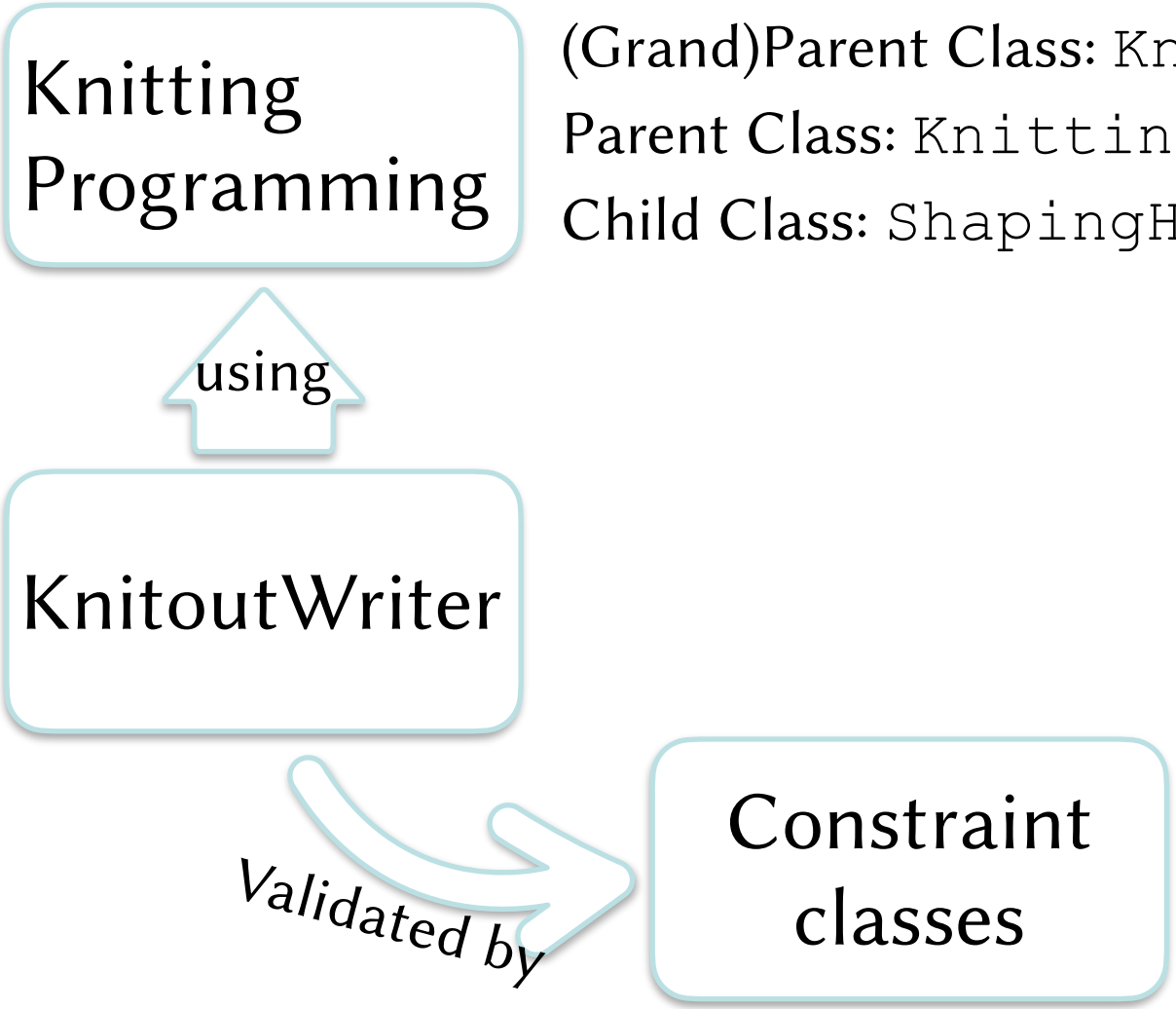
- a bit meta, but the knitout language we use to program the knitting machines has its own constraint (the syntax, just like how Python has its syntax)
- hard constraints, because not having the correct syntax means invalid knitout code, which could not be converted into the .dat format readable by KnitPaint

Machine Programming Constraint Handling

- In pattern programming:
  - if you are coding your own functions, use the KnitoutWriter class that you programmed yourself from HW5 which includes assertions for validation
  - you can also use known working knitout_helper functions from prior assignments

(Other Constraints)

Mental Model



(Grand)Parent Class: `KnitoutWriter`

Parent Class: `KnittingHelper`

Child Class: `ShapingHelper`

Parent Class: `KnittingHelper`

- contains all helper functions from the past HW2/3/4 `knitout_helpers.py`, but implemented using `KnitoutWriter`
  - all functionalities for flat knitting and two types of colorwork are included
- does not contain
  - plating
  - illusion knitting

Child Class: `ShapingHelper`

- inherits from `KnittingHelper` but implements additional helper functions for shaping
  - tubular cast on
  - increases
  - decreases
  - short-rows
- does not contain
  - cables
  - laces
  - shaping + colorwork/textures

Using `KnittingHelper` and `ShapingHelper` for your projects

- If your project involves shaping and tubular knitting:
  - treat this as additional, **completely optional** reference
  - these helpers are purely for making sure the suitable sequences of operations are being used to achieve a desired knitting output
    - *they do not decide the pattern, e.g., where the shaping should be placed*
    - you can rewrite them to suit your needs too! It's common that the function parameters and return values might not match what you expect but the same logic still applies

Exercises: let's open the Jupyter Notebook for today

- Try to implement a couple of functions in `KnittingHelper`
- We will go over at least one example for the `ShapingHelper`
- End with an example using the helper classes
- All the code will be updated within the project starter code and posted on Ed so that you can use them or modify them. If you have already started and coded a lot of these functionalities, that's totally fine and actually great! These code are only provided as a reference just in case you get stuck.