# Design Patterns Final: Initial specifications

1. System can handle four types of plastics:
   1. ABS
   2. Polypropylene
   3. Polyethelene
   4. PET
   5. If the plastic is unknown, default to ABS
      1. cout << " <>Unknown plastic |<unknown plastic>| defaulting to ABS.\n";
2. Handle orders up to 50,000:
   1. If no size specified, default to 100
   2. If size > 50,000, default to 50,000
      * 1. cout << " <>No size specified, defaulting to 100.\n";
3. Implement the following three Packagers:
   1. Bulk (just dump part to output bin)
   2. ShrinkWrap
   3. HardPack
   4. If Packager unknown, default to Bulk
      * 1. cout << " <>Unknown packager |<unknown packager>| defaulting to Bulk packager.\n";
4. Support three injection molding machines:
   1. IJM\_110 - good for aluminum molds with 1 cavity
   2. IJM\_120 - good for aluminum molds with 2 cavities
   3. IJM\_210 - good for steel molds with 1 cavity
5. Support two mold metals:
   1. Aluminum
   2. Stainless steel
6. Support molds with multiple cavities:
   1. 1 cavity
   2. 2 cavities
7. Support two conveyor belts:
   1. Linear
   2. Y-Split
8. Support two output bins:
   1. CardboardBox
   2. PalletBox
9. Setup the injection line based on run size and Packager:
   1. cout << " Setup injection line for <size> run with <Packager> packager:\n";
   2. size <= 10,000: cout << " IJM\_110 - Aluminum(1) - Linear conveyer belt - CardboardBox\n";
   3. size <= 20,000: cout << " IJM\_120 - Aluminum(2) - Y-Split conveyer belt - CardboardBox\n";
   4. size <= 50,000: cout << " IJM\_210 - Steel(1) - Linear conveyer belt - PalletBox\n";
10. Process order:
    1. Get mold
    2. Insert tags into mold
    3. Load plastic, color die, and additive bins
    4. Simulate cycling the IJM for the specified plastic <size> times
       1. Just once, specify the injection cycle (depends on plastic type)
       2. Just once, simulate a full parts bin asking upline machines to pause
    5. Clean the mold (ignore molds that are used up, clean them all)
11. Support three mold shapes with specified volumes in cc:
    1. Duck (35 cc)
    2. Car (40 cc)
    3. Hero (50 cc)
12. Molds are be pulled from inventory or milled from bulk metal
    1. (To keep the final simple, the order will specify the mold location.)
       1. If location and shape unknown, default to Duck from inventory
       2. cout << " <>Can't find place |<loc>| to get |<shape>| mold from, defaulting to duck from inventory.\n";
    2. Inventory:
       1. cout << " Pull <shape> mold from inventory.\n";
    3. Mill:
       1. cout << " Create <shape> mold from mill with <n> cavities:\n"
13. Shapes are milled with three techniques (actual steps simulated, see "couts" below):
    1. drill
    2. cut
    3. grind
14. Support two finishes:
    1. smooth
    2. rippled
15. Support two milling platforms:
    1. HighCarbon tools (suitable for aluminum)
       1. cout << " using HighCarbon tools (drill, cut, and high speed grind) to mill <metal> block into <n> <shape> shapes with <finish> finish.\n";
    2. Carbide tools (suitable for steel)
       1. cout << " using Carbide tools (high speed drill, cross cut, and layer grind) to mill <metal> block into <n> <shape> shapes with <finish> finish.\n";
16. Support three tags, widths in mm (inserted into molds, typically under the support base, think plastic toy):
    1. ModelNumber (2 mm)
    2. Country (2 mm)
    3. Date (2 mm)
    4. Total space for tags is 20 mm
       1. cout << " Insert tags [space separated list of tags ] of width <n>/20 mm.\n";
    5. Ignore unknown tags
       1. cout << "Ignoring unknown tag <tag>.\n"
17. Support two additives, the order specifies the volume in cc:
    1. UVInhibiter
    2. AntiBacterial
18. Support six colors (for simplicity, assume volume is 10% of shape, and die is independent of plastic type):
    1. black
    2. brown
    3. red
    4. orange
    5. yellow
    6. green
    7. If no color specified, default to black.
       1. cout << " <>No color specified, defaulting to black.\n";
19. Simulate mixing plastic, color, and additives with volume recipe & total (account for multiple cavities):
    1. cout << " Load plastic, color, and additive bins.\n;
    2. cout << " Recipe: <plastic>(<vol>) <color>(<vol>) <additive1>(<vol>) <additive2>(<vol>) Total = <vol>.\n";
20. Support three injection cycles, depends on plastic:
    1. ABS: heat to 440 - inject at 125 PSI - cool to 360 - progressive eject
    2. Poly: heat to 350 - inject at 90 PSI - cool to 290 - smooth eject
    3. PET: heat to 404 - inject at 110 PSI - cool to 340 - smooth eject
21. Support three upline machines pausing when package/output/parts bin is full:
    1. IJM
    2. Conveyer belt
    3. Packager
    4. Note: (The example output file is a little weak here, done more correctly, the diffs will be off a little.)
22. Simulate cycling the IJM for the specified plastic times
    1. cout << " Cycle IJM for <plastic> <size> times.\n";
    2. cout << " Close - heat to <temp> - inject at <psi> PSI - cool to <temp> - separate - <technique> eject.\n";
    3. cout << " <Packager> package bin full...\n";
       1. cout << " IJM pausing while CardboardBox package bin is swapped.\n";
       2. cout << " Conveyer belt pausing while CardboardBox package bin is swapped.\n";
       3. cout << " Packager pausing while CardboardBox package bin is swapped.\n";
23. Clean the molds, depends on plastic and metal:
    1. Use existing cleaning methods (see namespace legacy)
    2. cout << " Clean <plastic> <optional metal> mold: <cleaning steps>.\n";
24. Instrument the destructors:
    1. cout << "~<ClassName> ";
    2. Use newlines to logically group the class hierarchies
    3. Classes may be suffixed with design pattern name/abbreviation and/or presentation order
25. To get the diffs to zero:
    1. In main(int argc, char\* args[]):
       1. cout << "Hello DP4.\n" << endl;
       2. final\_design\_file::demo(string(args[1])); // Pass in the input file (orders\_1.txt).
       3. cout << "Aloha DP4.\n";