1. Mouse Circles

```python
from campy.graphics.gwindow import GWindow
from campy.graphics.gobjects import GOval
from campy.gui.events.mouse import onmouseclicked
import random

MIN_RADIUS = 10
MAX_RADIUS = 100
COLORS = ['red', 'green', 'blue', 'pink', 'orange', 'purple', 'yellow']

class MouseCircleGraphics:
    def __init__(self, width=500, height=300):
        self.window = GWindow(width, height)

        onmouseclicked(self.draw_random_circle)

    def draw_random_circle(self, event):
        click_x = event.x
        click_y = event.y
        radius = random.randint(MIN_RADIUS, MAX_RADIUS)
        circle_x = click_x - radius
        circle_y = click_y - radius
        random_circle = GOval(2 * radius, 2 * radius, x=circle_x, y=circle_y)
        random_circle.filled = True
        random_circle.fill_color = random.choice(COLORS)
        self.window.add(random_circle)

def main():
    # main is very simple in this case, we just create an instance
    # of the graphics window.
    mouse_circle_program = MouseCircleGraphics()

    if __name__ == '__main__':
        main()
```

2. Rubber Band Graphics

```python
from campy.graphics.gwindow import GWindow
from campy.graphics.gobjects import GLine
from campy.graphics.gtypes import GPoint
from campy.gui.events.mouse import onmouseclicked, onmousemoved

class RubberBandGraphics:
    def __init__(self, width=500, height=300):
        self.window = GWindow(width, height)

        self.current_line = None
```
onmouseclicked(self.create_new_line)
onmousemoved(self.update_current_line_endpoint)

def create_new_line(self, event):
    line_x = event.x
    line_y = event.y
    self.current_line = GLine(line_x, line_y, line_x, line_y)
    self.window.add(self.current_line)

def update_current_line_endpoint(self, event):
    if self.current_line != None:
        new_x = event.x
        new_y = event.y
        new_endpoint = GPoint(new_x, new_y)
        self.current_line.end = new_endpoint

def main():
    rubber_band_program = RubberBandGraphics()

    if __name__ == '__main__':
        main()

3. Employee Tracker

class Employee:
    def __init__(self, name, job_title):
        self.name = name
        self.job_title = job_title
        self.years_of_service = 0
        self.salary = 0

        def promote(self, threshold):
            if self.years_of_service >= threshold:
                self.salary *= 2
                self.job_title = "Senior " + self.job_title

        def get_name(self):
            return self.name

        def get_job_title(self):
            return self.job_title

        def get_salary(self):
            return self.salary

        def get_years_of_service(self):
            return self.years_of_service

        def set_job_title(self, job_title):
            self.job_title = job_title

Portions of this handout based on work by Eric Roberts, Sam Redmond, and Brahm Capoor
def set_salary(self, salary):
    self.salary = salary

def set_years_of_service(self, years_of_service):
    self.years_of_service = years_of_service

def main():
    employees = []
    while True:
        name = input('Name: ')
        if name == '':
            break
        title = input('Title: ')
        salary = int(input('Salary ($) : '))
        years_of_service = int(input('Years of service: '))
        employee = Employee(name, title)
        employee.set_salary(salary)
        employee.set_years_of_service(years_of_service)
        employees.append(employee)

        # promote all employees that have worked here for more than 5 years
        for employee in employees:
            employee.promote(threshold=5)

        # print all employee information
        for employee in employees:
            print('--- ' + employee.get_name() + ' (' + employee.get_job_title() + ')
        print('Length of service (years): ' + str(employee.get_years_of_service()))
        print('Salary: $' + str(employee.get_salary()))

if __name__ == '__main__':
    main()

4. Up, Up, and Away!

class Airport:
    def __init__(self):
        self.planes = []
        self.plane_id = 1

    def build_plane(self):
        print("Airport log: Building new plane")
        new_plane = Airplane(self.plane_id)
        self.planes.append(new_plane)
        self.plane_id += 1

    def dispatch_plane(self):
        first_plane = self.planes[0]
if first_plane.is_airborne():
    print("Airport log: ERROR - plane already airborne")
else:
    first_plane.take_off()
    # now that it is guaranteed to be in the air, we can safely
    # remove it from the airport
    self.planes.pop(0)

def num_planes_left(self):
    return len(self.planes)

class Airplane:
    def __init__(self, id_num):
        self.id = id_num
        self.airborne = False

        self.build_fuselage()
        self.build_wings()

    def take_off(self):
        print("Airplane log: Plane #" + str(self.id) + " taking off!")
        self.airborne = True

    def build_fuselage(self):
        print("Airplane log: Built fuselage for plane #" + str(self.id))

    def build_wings(self):
        print("Airplane log: Built wings for plane #" + str(self.id))

    def is_airborne(self):
        return self.airborne

def main():
    lax = Airport()

    # build three planes
    for i in range(3):
        lax.buildPlane()

    # dispatch the first two planes
    for i in range(2):
        lax.dispatch_plane()

    # build one last plane
    lax.build_plane()

    # tell all planes to depart
    while lax.num_planes_left() > 0:
        lax.dispatch_plane()

if __name__ == '__main__':
    main()